Título: Transformación de modelos del estándar de salud HL7 a UML/OCL

Autor: David Ortiz López

Fecha: Junio de 2011

Director: Antoni Olivé Ramon

Departamento del director: Departamento de Ingeniería de Servicios y Sistemas de Información

Codirector: Antonio Villegas Niño

Departamento del codirector: Departamento de Ingeniería de Servicios y Sistemas de Información

Titulación: Ingeniería Informática

Centro: Facultat d'Informàtica de Barcelona (FIB)

Universidad: Universitat Politècnica de Catalunya (UPC),

BarcelonaTech

INTRODUCCIÓN

MODELIZACIÓN CONCEPTUAL

HL7

Índice de figuras

Índice de tablas

Índice de ejemplos

INTRODUCCIÓN

1 DESCRIPCIÓN DEL PROYECTO

En este primer capítulo se describen los aspectos más generales del proyecto desarrollado así como los de este documento.

1.1. Acerca del proyecto

"Transformación de modelos del estándar de salud HL7 a UML/OCL" es un proyecto final de carrera realizado por David Ortiz López, estudiante de ingeniería informática en la *Facultat d'Informàtica de Barcelona* (FIB), perteneciente a la *Universitat Politècnica de Catalunya* (UPC).

En el presente documento, se describe todo el trabajo desarrollado durante el transcurso del proyecto.

1.2. Motivación

Las asignaturas que me resultaron más interesantes durante mis estudios, fueron aquellas relacionadas con la ingeniería del software y la modelización conceptual. Por ello, me dirigí a Antoni Olivé Ramon, profesor de la asignatura de ingeniería de requisitos, para ver si podíamos iniciar un proyecto relacionado con esa temática.

Él, junto al codirector de este proyecto, Antonio Villegas Niño, estudiante de doctorado de su mismo grupo de investigación y que ya había trabajado con el estándar HL7 anteriormente, me asignaron el presente proyecto, que me permitiría poner en práctica a la vez que expandir, los conocimientos adquiridos en mis principales áreas de interés.

1.3. Objetivo del proyecto

El estándar *Health Level Seven* (HL7) define cómo tienen que ser las interacciones entre distintas entidades dentro del dominio de la salud. Este estándar contiene múltiples esquemas conceptuales que definen toda la información necesaria para especificar un amplio abanico de mensajes y documentos clínicos tales como una orden de laboratorio, una alta médica o un informe de radiología.

Los esquemas conceptuales que se pueden encontrar en el estándar HL7, están descritos utilizando un lenguaje gráfico de modelización propio, hecho que dificulta el trabajo de todas aquellas personas interesadas en su utilización.

El objetivo del presente proyecto, consiste en construir una herramienta, que de forma automática, permita transformar esos esquemas en otros descritos mediante UML y OCL y que contengan el mismo conocimiento.

1.4. Justificación del objetivo del proyecto

Bajo nuestro punto de vista, la adopción de un lenguaje de modelización propio no constituye la mejor opción. Esta elección conlleva problemas nada despreciables, como son la necesidad de proporcionar una formación específica a todas aquellas personas que deseen trabajar con el estándar, o el tener que diseñar herramientas especializadas para poder dar soporte a sus actividades de desarrollo e implementación.

UML y OCL son los estándares que a día de hoy cuentan con una aceptación más generalizada en el campo de la modelización conceptual. Su adopción por parte de la comunidad de HL7, conllevaría principalmente dos ventajas muy significativas. La primera, es que no sería necesario formar en estándares propios a los colaboradores, puesto que como ya se ha comentado, tanto UML como OCL son ampliamente conocidos por especialistas en sistemas de información e ingeniería del software. La segunda, es que se podría utilizar una amplia gama de herramientas ya disponibles directamente sobre los esquemas del estándar HL7. Entre estas herramientas podemos encontrar generadores de código, validadores de modelos, editores gráficos, etc.

Incluso desde HL7, algunos miembros piden usar UML para representar los esquemas conceptuales del estándar. En una de las listas de correo de HL7, René Spronk, uno de los representantes de la división holandesa del estándar, afirmaba lo siguiente: "*I'd personally suggest that it's probably time that HL7 created (and publishes) an UML (enriched with OCL) specification of its static model artefacts as well as the data types. Having UML allows implementers to use tons of standard tools*".

1.5. Acerca de este documento

En este documento, se pretende exponer todo lo relacionado con la construcción de la herramienta de transformación, además de proveer a los lectores que no son expertos en la materia de toda la teoría tanto de modelización conceptual como de HL7, que resulta imprescindible para entender el proceso de desarrollo. El resto de capítulos del documento se estructuran en una serie de bloques tal y como se explica a continuación, con el objetivo de facilitar su lectura.

En el bloque "Modelización conceptual", que comprende los capítulos del 2 al 6, se explica toda la teoría de modelización conceptual relacionada con el proyecto. Se exponen algunas de las características principales de UML, OCL y XMI, y se proporciona una introducción al metamodelado.

En el bloque "HL7", que comprende los capítulos del 7 al 11, se presentan todos aquellos aspectos del estándar HL7 con los que se debe estar familiarizado para entender el desarrollo de la herramienta de conversión desarrollada.

En el bloque "Herramienta de transformación de modelos", que comprende los capítulos del 12 al 23, se exponen las distintas fases de desarrollo de la herramienta de transformación de forma detallada, así como los resultados obtenidos y los problemas encontrados.

En el bloque "Planificación y costes", que comprende los capítulos 24 y 25, se muestra la planificación temporal del proyecto así como un estudio de su coste.

En el bloque "Conclusiones", que contiene el capítulo 26, se exponen las conclusiones extraídas del trabajo realizado.

Para finalizar, las últimas páginas se completan con una bibliografía en la que se pueden encontrar enlaces web, libros y artículos de interés.

1.6. Agradecimientos

Antes de dar paso al resto de capítulos de este documento, me gustaría agradecer a varias personas su colaboración en el proyecto.

En primer lugar, quiero agradecer a Antoni Olivé el haberme brindado la oportunidad de realizar este proyecto y el haber resuelto algunas de las dudas que surgieron durante el transcurso del mismo.

También quisiera dar las gracias a Antonio Villegas por su enorme grado de involucración en el proyecto desde el primer momento y por todas sus sugerencias y correcciones que sin lugar a dudas, han contribuido a que se hayan podido cumplir los objetivos marcados.

Por último, me gustaría agradecer a Diego Kaminker que se reuniese con nosotros para poder pedirle su opinión sobre el trabajo realizado.

MODELIZACIÓN CONCEPTUAL

2 INTRODUCCIÓN A LA MODELIZACIÓN CONCEPTUAL

Para comprender las explicaciones expuestas en el bloque dedicado al desarrollo de la herramienta de transformación de modelos que se ha construido en el proyecto que nos ocupa, resulta necesario conocer toda una serie de aspectos relacionados con la modelización conceptual.

El objetivo del bloque que se inicia con este capítulo, consiste en presentar todos esos aspectos. Así, se presenta una introducción a la modelización conceptual y los esquemas conceptuales, se exponen algunas de las características principales de UML y OCL, se proporciona una breve introducción al metamodelado explicando el metamodelo de UML y finalmente, se introducen las especificaciones escritas en XMI.

2.1 Esquemas conceptuales

En el ámbito de la disciplina de la ingeniería del software, se denomina modelización conceptual, a la actividad consistente en especificar y describir el conocimiento general que un sistema de información necesita conocer para poder realizar sus funciones correctamente.

La modelización conceptual es una actividad estrictamente necesaria en cualquier proceso de desarrollo de un sistema software. Su principal objetivo, consiste en definir el esquema conceptual de ese sistema concreto.

Contrariamente a lo que se piensa en algunas ocasiones, cualquier sistema software cuenta con un esquema conceptual. Asociar los esquemas conceptuales a documentos físicos, puede conducir a pensar lo contrario, no obstante, un esquema conceptual no tiene por qué estar escrito, puede estar en la mente de los diseñadores y programadores de dicho software. Lo que está claro, es que sin un esquema conceptual, el sistema software resultante carecerá del conocimiento que le resulta necesario para ejecutar sus funciones con éxito y por consiguiente, no será de ninguna utilidad y todos los esfuerzos dedicados en su desarrollo habrán sido infructuosos.

Los esquemas conceptuales contienen aquellos conceptos relevantes para el sistema de información diseñado, también llamados entidades, así como las relaciones que se establecen entre pares de esos conceptos, conocidas como asociaciones.

Los lenguajes que permiten definir esquemas conceptuales, se conocen con el nombre de lenguajes de modelización conceptual. En el proyecto que nos ocupa, los lenguajes de modelización conceptual involucrados son el propio de *Health Level* 7 y el *Unified Modeling Language* (UML). En el siguiente capítulo, se explican los aspectos más importantes de UML. Por otro lado, al lenguaje de modelización propio de HL7 se le ha dedicado un bloque entero.

Resulta necesario hacer mención al hecho que, en los documentos que incluye el estándar HL7, se denomina modelos de información a todos aquellos artefactos que cumplen la definición de esquema conceptual proporcionada anteriormente. Por este motivo, en el bloque dedicado a HL7, se usarán las dos denominaciones indistintamente.

3 UNIFIED MODELING LANGUAGE (UML)

UML es, a día de hoy, el lenguaje de modelización conceptual por antonomasia. Además de ser el más conocido y utilizado dentro de su campo, es el que se impulsa desde la *Object Management Group* (OMG), organización que se encarga de promover estándares relacionados con los sistemas orientados a objetos.

En este capítulo, se detallan aquellos aspectos de UML que resultan relevantes para comprender los resultados arrojados por el software transformador que se ha desarrollado.

En concreto, se describen algunos de los elementos que componen UML. Presentarlos todos, queda lejos del alcance de este documento, puesto que el estándar UML es muy extenso y en el presente proyecto, sólo nos interesa analizar un pequeño subconjunto de todo lo que presenta, aquellos elementos que intervienen en el proceso de transformación de los esquemas conceptuales de HL7. El lector que desee profundizar en las características de UML, puede encontrar su especificación formal completa en [UmlEsp].

3.1 Entidad

Una entidad se define como un concepto cuyas instancias en un momento determinado, corresponden a objetos existentes dentro del dominio que se está tratando.

Imaginemos que estuviésemos definiendo el esquema conceptual de un sistema de información cuyo objetivo fuese facilitar a los alumnos de una determinada universidad su proceso de matriculación. En este dominio, resulta evidente que deben intervenir entidades como alumno o asignatura. En este contexto, deberíamos modelar ese par de conceptos, entre muchos otros y para llevar a cabo dicha tarea, podríamos emplear UML.

Tal y como se puede apreciar en la figura 3.1, el resultado de modelar esos dos conceptos utilizando UML, resulta sencillo de interpretar. Cada uno es representado mediante un rectángulo con tres secciones. En la superior, encontramos el nombre del concepto modelado y en la segunda, todos los atributos que posee dicho concepto junto a sus tipos de datos (los tipos básicos de UML son cuatro: *string*, *boolean*, *integer* y *real*). En ocasiones, se puede encontrar una tercera división, siempre situada por debajo de la dedicada a los atributos, en la que tienen cabida todas las operaciones asociadas a ese concepto concreto. En UML, al rectángulo entero se le concede el nombre de clase UML.

En el ejemplo mostrado, se puede ver que los atributos de la clase *Alumno* son: *identificador*, *nombre*, *apellidos*, *dirección*, *teléfono* y *email*, todos ellos de tipo *string*. Además, esta clase presenta una operación llamada *matricular*. Por otro lado, la clase *Asignatura* presenta tres atributos: *código*, *nombre* y *troncal*.

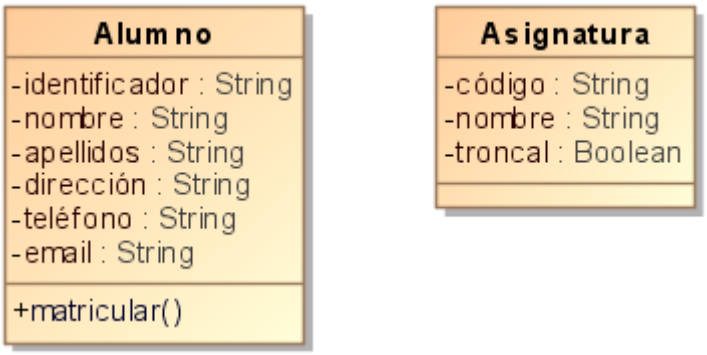


Figura 3.1 Clases UML

3.2 Asociación

UML permite expresar asociaciones que se dan entre dos entidades representadas como clases. Una asociación, se representa mediante una línea cuyos dos extremos son las clases que asocia. En cada uno de esos extremos, encontramos una multiplicidad que nos indica el número de instancias de esa clase que pueden participar en la asociación. Además, en ocasiones también encontramos nombres de rol, aunque estos últimos sólo son estrictamente necesarios cuando existe más de una asociación que une el mismo par de clases, debido a que esta situación da lugar a la ambigüedad.

En el ejemplo de la figura 3.2, encontramos dos relaciones. La primera, *cursa*, relaciona la clase *Alumno* con *Titulación* e indica que un alumno puede cursar una o más titulaciones y que una titulación puede tener cualquier número de alumnos inscritos. La segunda, *imparte*, relaciona la clase *Titulación* con *Asignatura* y representa que una titulación puede impartir una o más asignaturas, mientras que una asignatura determinada, sólo puede pertenecer a una titulación.

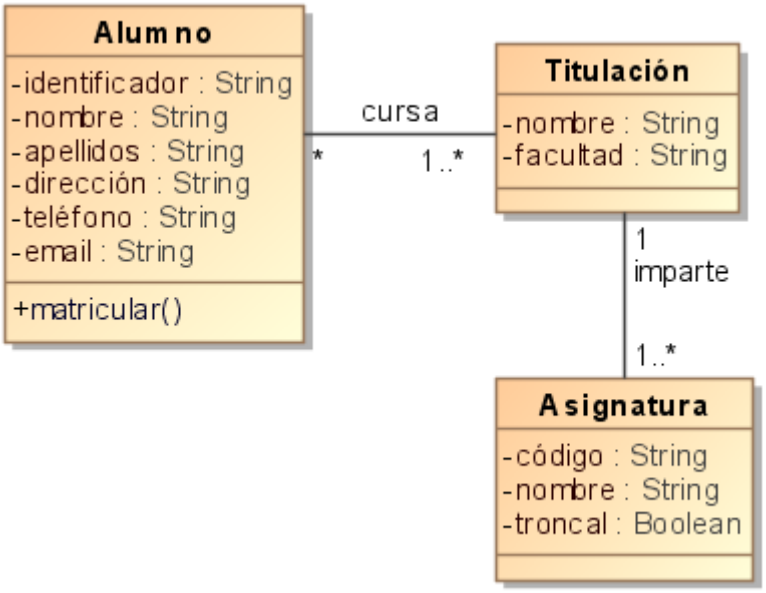


Figura 3.2 Asociaciones UML

3.3 Generalización

Una generalización es una asociación de clasificación que se da entre dos clases UML. Una de ellas modela un concepto de forma general, y la otra lo hace de forma más específica. Esta última contiene todos los atributos, asociaciones y restricciones de la general, además de información adicional. A la clase general, normalmente se la conoce como clase padre, y a la específica como clase hija. También se suele decir que esta última hereda de la clase general.

En UML, las generalizaciones se representan mediante una flecha que va desde la clase específica hasta la general. En la figura 3.3, se muestra un ejemplo que podría corresponder a cualquier sistema de información que involucrase vehículos. En él, la clase *Vehículo* es la general, y *Coche* y *Camión* constituyen las específicas. Como se puede observar, estos dos conceptos, a pesar de ser considerados vehículos, presentan información diferente. En el primero, nos interesa conocer el número de plazas, mientras que en el segundo se necesita la tara. No obstante, ambos pertenecen a la familia de vehículos y por tanto, contienen información común como el precio y el número de licencia.

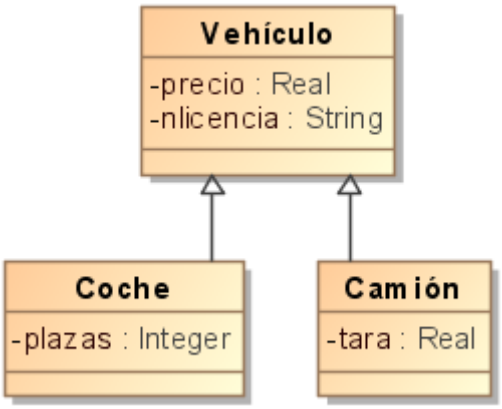


Figura 3.3 Generalizaciones UML

3.4 Package

Los *packages* son un mecanismo del que dispone UML para poder agrupar elementos de una forma comprensible. Su uso no comporta ningún impacto semántico en los esquemas conceptuales, esto es, el conocimiento que contiene un esquema determinado no cambia al utilizar packages, simplemente facilita su interpretación.

En UML, los *packages* aparecen representados mediante una figura con forma de carpeta. También se permite importar elementos y *packages* enteros externos. Este hecho, se representa mediante una flecha discontinua que va desde el *package* en cuestión hasta el elemento importado.

En la figura 3.4, se muestra un ejemplo del uso de packages en UML. En este caso concreto, se tiene un esquema referente a un sistema de matriculación universitaria. Con el fin de poder presentar el esquema de una forma más ordenada, se podría optar por representar todas las asignaturas dentro de un *package* llamado *Asignaturas* y que fuese importado por el principal.

Figura 3.4 Packages UML

3.5 Estereotipo

En algunas ocasiones, resulta deseable ampliar los mecanismos que proporciona UML para poder adaptarlo a las necesidades del dominio que se está modelando. Una de las soluciones que proporciona UML para este tipo de casos, es el uso de estereotipos.

En UML, un estereotipo es una clase que normalmente se define con el objetivo de poder extender las características que posee un elemento perteneciente a un modelo determinado. Aunque no siempre es así, a veces se utilizan simplemente como un elemento diferenciador, para agrupar los elementos de un mismo tipo en distintas categorías.

En las clases en las que se aplica un estereotipo, éste aparece representado encima del nombre entre las cadenas de caracteres '<<' y '>>'.

En la figura 3.5, se puede apreciar un ejemplo del primero de los usos de estereotipos señalados anteriormente. En este caso concreto, se extiende la clase *Class* del metamodelo UML para poder reflejar en los esquemas, información que por defecto UML no define en esa clase, por ejemplo, el autor y la fecha de creación.

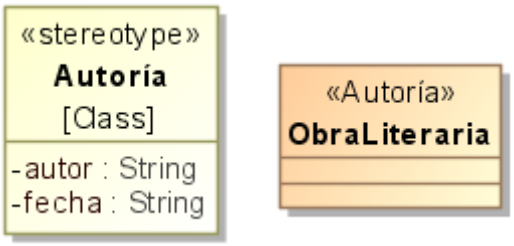


Figura 3.5 Estereotipo UML

En la figura 3.6, se encuentra un ejemplo de uso de estereotipos utilizados como elemento diferenciador. En algunos patrones de ingeniería del software, se distingue entre clases de tipo *boundary* (vistas con las que el usuario puede interactuar), *control* (tipo que opera con los elementos del dominio) y *entity* (elementos del dominio). Así, con el objetivo de hacer los esquemas más entendedores, esos tres tipos de clases se suelen estereotipar.

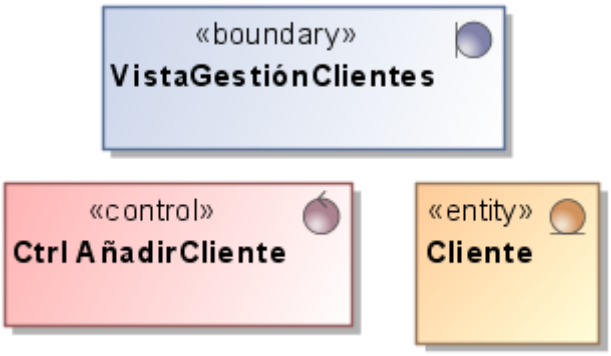


Figura 3.6 Estereotipos UML como elemento diferenciador

3.6 Comentario

Los comentarios son anotaciones o aclaraciones que hacen referencia a un elemento o un conjunto de elementos de un determinado esquema. Al igual que ocurre con el tema de los *packages* comentados en este mismo capítulo, los comentarios no tienen ningún impacto semántico en el esquema, simplemente proporcionan textos redactados de forma arbitraria, con el objetivo de aclarar cuestiones que puedan resultar ambiguas o comentar aspectos entre el equipo de trabajo, con la misma función que realizan los comentarios del código en los lenguajes de programación. En la figura 3.7, encontramos un ejemplo de comentario ejerciendo esa última función.

Tal y como se puede apreciar en ese mismo ejemplo, en UML los comentarios se representan mediante rectángulos con la esquina superior derecha doblada, unidos al elemento o conjunto de elementos a los que hacen referencia mediante una línea discontinua.

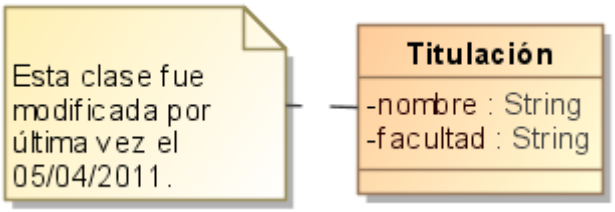


Figura 3.7 Comentario UML

4 OBJECT CONSTRAINT LANGUAGE (OCL)

Debido a las limitaciones de UML, los esquemas conceptuales representados mediante este lenguaje, pueden ser incompletos o dar pie a la ambigüedad. Esto es debido a que, normalmente, no todas las restricciones que debe cumplir un determinado esquema conceptual pueden ser expresadas de forma gráfica mediante los mecanismos proporcionados por UML. Por este motivo, sin emplear ningún lenguaje auxiliar, la única solución posible pasa por definir esas restricciones en lenguaje natural, lo que conduce irremediablemente a ambigüedades.

En el ejemplo que se muestra en la figura 4.1, queda reflejada esa situación. UML por sí solo no permite expresar de forma gráfica las siguientes restricciones: “ninguna persona menor de edad puede estar casada” o “una persona no puede tener esposo y esposa simultáneamente”.

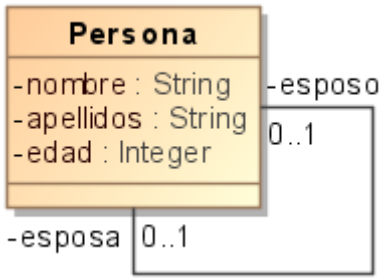


Figura 4.1 Modelización en UML de los conceptos persona y matrimonio

Esta situación provoca la necesidad de disponer de un lenguaje que permita definir restricciones que no pueden ser expresadas de forma gráfica y que además, lo pueda hacer de forma no ambigua. Los lenguajes que presentan esta última característica se denominan formales.

El lenguaje formal que hoy en día goza de una mayor aceptación para solventar ese tipo de carencias presentes en UML, es el *Object Constraint Language* (OCL). Cabe destacar, que además de utilizarse para especificar restricciones, se emplea con un amplio abanico de fines distintos, como pueden ser la definición de precondiciones y postcondiciones de operaciones, especificación de *queries*, etc. Sin embargo, el único propósito con el que será utilizado en este proyecto es el de definir restricciones textuales.

Por este motivo, en el presente documento no se ahonda en las características de OCL y simplemente se introduce brevemente y se presentan aquellos aspectos que resultan

esenciales para comprender la definición de restricciones mediante pequeños ejemplos. Para consultar la especificación completa de OCL, se remite al lector interesado a [OclEsp].

Así, en OCL las restricciones comentadas sobre ejemplo mostrado en la figura 4.1 se definirían tal y como se describe a continuación:

“Ninguna persona menor de edad puede estar casada”:

```
Context Persona inv:
  self.esposa->notEmpty() implies self.esposa.edad >= 18 and
  self.esposo->notEmpty() implies self.esposo.edad >= 18
```

Ejemplo 4.1 Restricción OCL "Ninguna Persona menor de edad puede estar casada"

“Una persona no puede tener esposa y esposo al mismo tiempo”:

```
Context Persona inv:
  not (self.esposo->notEmpty() and self.esposa->notEmpty())
```

Ejemplo 4.2 Restricción OCL "Una persona no puede tener esposa y esposo al mismo tiempo"

A continuación, tal y como se había avanzado, se procede a detallar los aspectos más relevantes de OCL a partir de ese par de reglas ejemplificadas.

Cada expresión OCL, se define en el contexto de una instancia de un tipo determinado. Esto se indica mediante la palabra clave *context*. En los ejemplos, podemos ver que aparece *context Persona*. Esto significa que, en nuestro caso, las expresiones definidas tienen que ser ciertas para todas las instancias de la clase *Persona*.

La otra palabra clave que resulta imprescindible para comprender las expresiones OCL, es *self*. Ésta se refiere a la instancia que se está tratando en ese momento. Las reglas que se encuentran definidas dentro del contexto de la clase *Persona*, como las mostradas en los ejemplos, se comprueban para cada una de las instancias de dicha clase. Así, *self* se referirá a la instancia de *Persona* contra la que la regla se está validando en ese momento.

A partir de una instancia concreta, OCL permite conocer el valor que toman sus atributos y también, navegar a través de sus asociaciones. De este modo, en los ejemplos anteriormente expuestos, *self.nombre* hace referencia al atributo *nombre* de la instancia de *Persona* que se esté tratando en ese momento y *self.esposo* a la instancia de *Persona* que está relacionada con ella y que juega el rol de *esposo* en la asociación.

Cabe destacar que, cuando se navega por las asociaciones de una clase determinada, no se obtiene una sola instancia de la clase con la que se relaciona, sino que se obtiene una colección. En nuestro caso, al ser las multiplicidades de los roles *esposo* y *esposa* 0..1, las colecciones contendrán a lo sumo un único elemento, pero frecuentemente se da el caso en el que se tiene un número indeterminado de ellos.

Para tratar las colecciones, OCL proporciona un amplio abanico de funciones. En el par de ejemplos expuestos, se puede encontrar dos de ellas, *size()*, que devuelve el número de elementos contenidos en una colección, y *notEmpty()*, que retorna un booleano indicando si la colección es vacía.

Como se ha afirmado anteriormente y se ha podido comprobar mediante el caso ejemplificado, la utilización única de UML produce esquemas incompletos o ambiguos. Por otro lado, las restricciones textuales descritas utilizando OCL, necesitan elementos UML que las doten de significado. Por esto, son lenguajes que se complementan, no oponen, y constituyen un tándem idóneo para la definición completa y correcta de esquemas conceptuales.

5 METAMODELADO

El metamodelado, es la actividad enmarcada dentro de la ingeniería del software que produce, principalmente, metamodelos.

Los metamodelos se definen como un conjunto de esquemas y restricciones que constituyen una precisa definición de las reglas necesarias para crear modelos. En el proyecto que nos ocupa, se trabaja con los metamodelos de UML y HL7. El primero, será explicado a continuación, mientras que el de HL7 se presentará en el capítulo 13 de este documento.

Del mismo modo que los esquemas conceptuales definen el conocimiento necesario de que un sistema software necesita disponer para realizar sus funciones, un metamodelo define el conocimiento de un esquema conceptual.

Así, los esquemas conceptuales representan una abstracción de los conceptos del mundo real y los metamodelos, se encuentran en un nivel por encima de abstracción, tal y como se muestra en el esquema de la figura 5.1.

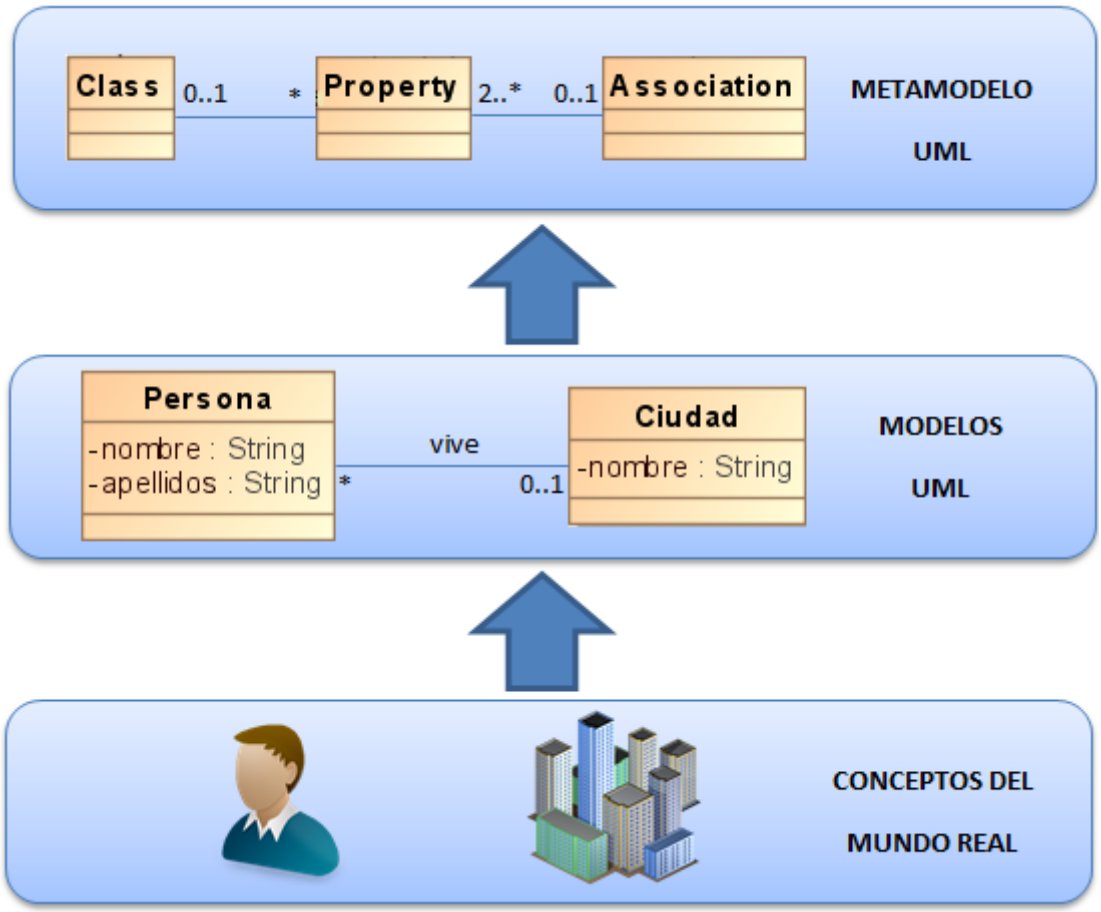


Figura 5.1 Niveles de representación

5.1 El metamodelo de UML

El metamodelo de UML es un modelo que representa el conocimiento general de todos los modelos definidos en UML. Dicho metamodelo, ha sido diseñado por la *Object Management Group* (OMG).

El resultado de las transformaciones del software desarrollado en el proyecto que nos ocupa, debe cumplir lo definido en el estándar de UML, o dicho de otra manera, los esquemas resultantes deben ser instancias del metamodelo UML. Por este motivo, para poder comprender los capítulos dedicados a la explicación del desarrollo del software en sí, resulta imprescindible estar familiarizado con algunos fragmentos de dicho metamodelo.

El metamodelo de UML es muy extenso y presentarlo en su totalidad está lejos del alcance de este proyecto. Para justificar los resultados del software transformador, sólo resulta necesario analizar un subconjunto, que es el que se cubrirá con detalle en esta sección. No obstante, el lector que desee consultar su versión completa, la puede encontrar en [UmlEsp].

Con el objetivo de presentar los distintos elementos que componen el metamodelo de UML de una forma que resulten más fáciles de entender, se ha optado por presentarlos en subconjuntos de tamaño reducido y en los casos que se ha considerado oportuno, se acompañan de ejemplos. En ellos, se omiten elementos como atributos con el fin que resulten más claros. En algunos de ellos, aparecen instancias de clases o metaclases. Se debe tener en cuenta que éstas, en UML se representan igual que las clases pero con el carácter ":" precediendo a sus nombres.

Además, se ha decidido no mostrar las restricciones OCL que hacen referencia a cada uno de los diagramas mostrados, puesto que el objetivo de esta sección no es presentarlos de una manera completamente formal, sino proveer al lector del material teórico necesario para poder comprender las transformaciones de los esquemas HL7 que se tratarán en un bloque posterior.

Antes de proceder a explicar cada uno de esos fragmentos, debemos clarificar el significado del término metaclase, puesto que se hará referencia a él continuamente, debido a que todas las clases que aparecen en un metamodelo se denominan así.

Una metaclase es una clase cuyas instancias son clases. Esta definición puede resultar un tanto confusa a priori, pero si se piensa en la definición dada anteriormente para el término metamodelo, resulta lógico que todas las clases pertenecientes a un metamodelo sean

metaclases, puesto que un metamodelo no es más que un modelo cuyas instancias son modelos.

5.1.1 Elementos

En la figura 5.2, se presenta la metaclase *Element* y algunas de sus subclases principales. La mayoría de metaclases del metamodelo heredan de alguna de ellas.

Element es una metaclase abstracta sin superclases. Se usa como superclase de todas las metaclases que aparecen en el metamodelo de UML. Por otro lado, cualquier *Element* puede tener asociado uno o varios comentarios, que son instancias de la metaclase *Comment*.

Además, se puede encontrar la metaclase *NamedElement,* que representa elementos que pueden tener un nombre. Por su parte, *TypedElement*, tal y como su nombre indica, representa elementos que pueden tener un nombre y tipo asignados.

Como se ha comentado anteriormente, la mayoría de las metaclases del metamodelo son subtipos de alguna de las que se muestran en el fragmento del diagrama de la figura 5.2. Teniendo este hecho presente, con el objetivo de no complicar de forma innecesaria los diagramas que se muestran a continuación en este capítulo, no se volverán a mostrar las asociaciones binarias ni jerárquicas mostradas en él.

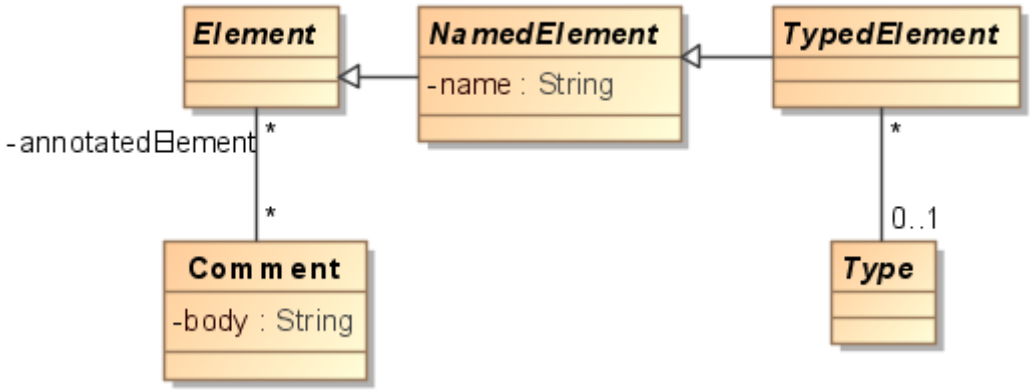


Figura 5.2 Elementos del metamodelo UML

5.1.2 Clases y asociaciones

En la figura 5.3, se presenta el fragmento del metamodelo referente a las clases y las asociaciones en UML. La metaclase *Class*, tiene como instancias a clases UML y está

relacionada con *Property*. Esta última clase puede jugar dos roles diferenciados, por un lado puede representar los atributos de una clase (asociación entre *Class* y *Property*), y por otro, los miembros de los extremos de una asociación UML (asociación entre *Property* y *Association*).

La asociación existente entre *Property* y *ValueSpecification* permite definir el valor por defecto de un atributo de una clase en caso que tenga. Por otro lado, la relación de herencia desde *Property* a *MultiplicityElement*, es la que permite definir la multiplicidad tanto de los atributos de una clase, como la de los extremos de una asociación.

Finalmente, cabe destacar el hecho que *Property* herede de *TypedElement*, lo cual permite que se puedan tener clases como extremos de una asociación cualquiera, ya que recordemos que *Class* es subtipo de *TypedElement*. Para terminar, por cada tipo de dato existente, habrá una instancia de la metaclase *DataType*.

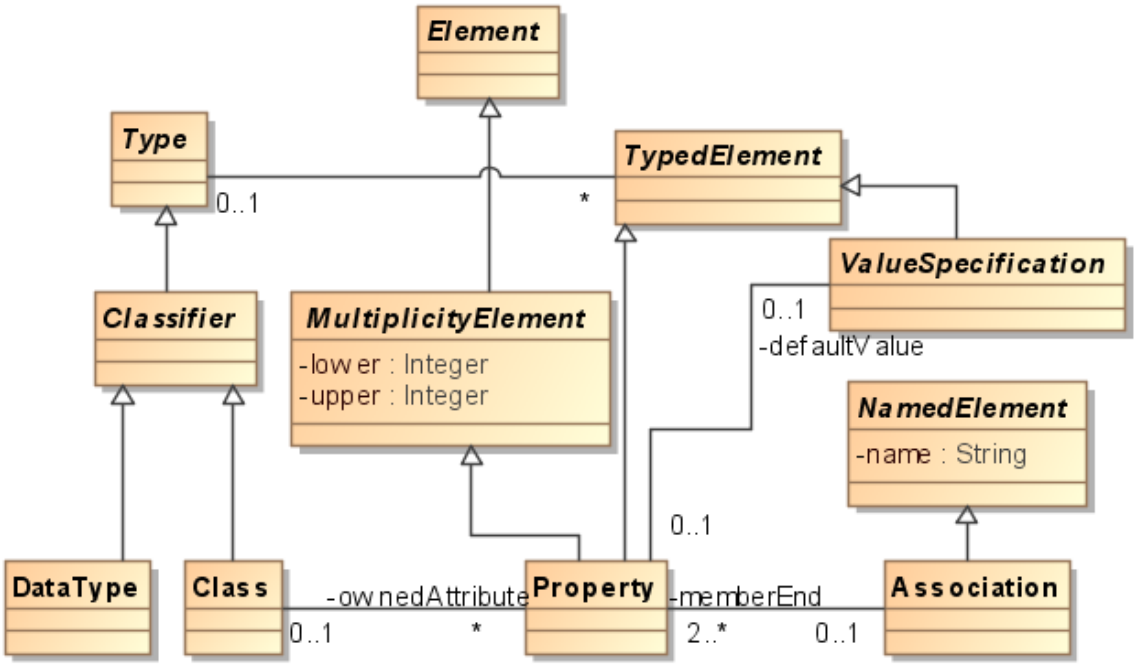


Figura 5.3 Clase y asociación en el metamodelo de UML

En la figura 5.4, se muestra un ejemplo de instanciación que involucra asociaciones. En él, el atributo *name* de la instancia de la metaclase *Association*, toma por valor el nombre que se le asigna a la asociación en el esquema citado, es decir, *cursa*. Además, esta instancia se relaciona con dos instancias de *Property*, cuyos atributos *name* toman el valor de los nombres de rol que contiene la asociación. Finalmente, estas dos instancias se relacionan con una

instancia de las clases Alumno y Titulación respectivamente, que son las que indican los tipos de los extremos de la asociación.

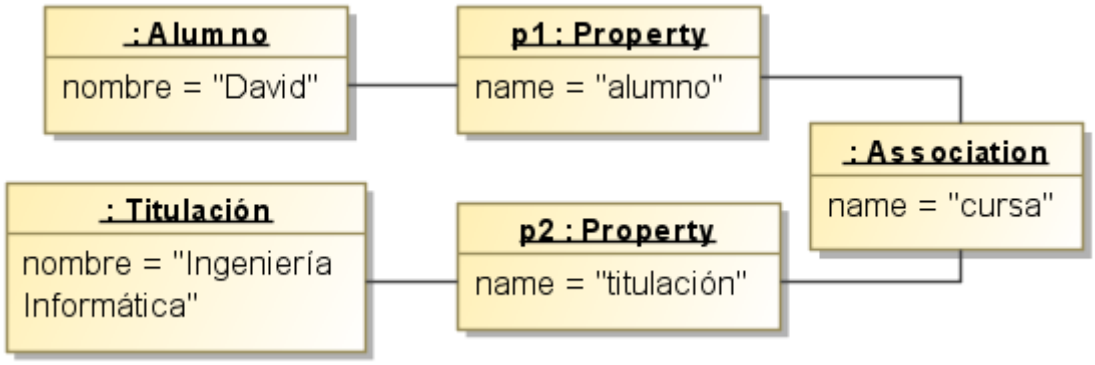


Figura 5.4 Instanciación de clases y asociaciones del metamodelo de UML

5.1.3 Generalizaciones

En la figura 5.5, se puede comprobar cómo se representa en el metamodelo de UML el hecho que este lenguaje permita construir una jerarquía de clases mediante especialización y generalización. La metaclase *Generalization*, tiene dos asociaciones con la metaclase *Classifier*. Como se puede deducir a partir de lo explicado anteriormente, una de ellas representa el concepto general dentro de la generalización, y el otro el más específico. Cabe destacar que, tanto las metaclases *Class* como *DataType*, son subtipos de *Classifier* y que por tanto, UML no sólo permite jerarquías de clases, sino también de tipos de datos.

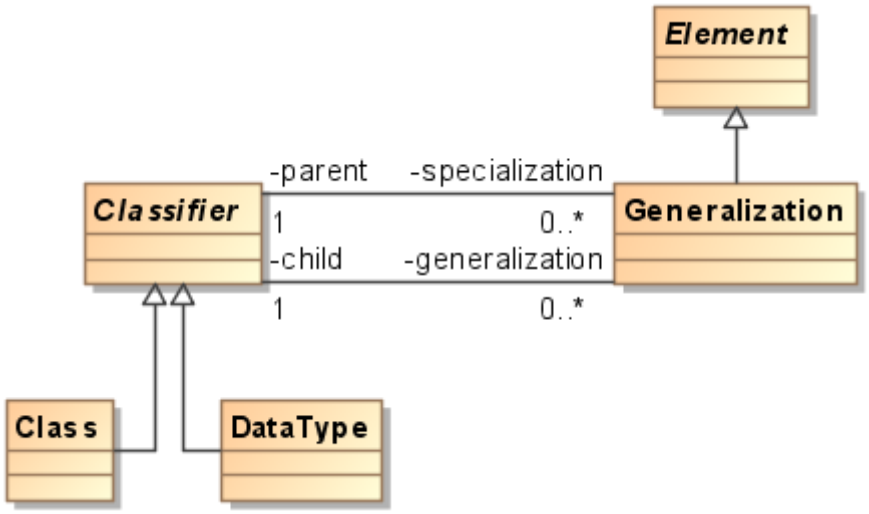


Figura 5.5 Generalización en el metamodelo de UML

En la figura 5.6, se muestra una instanciación de una generalización, concretamente la de la mostrada anteriormente en la figura 3.3.

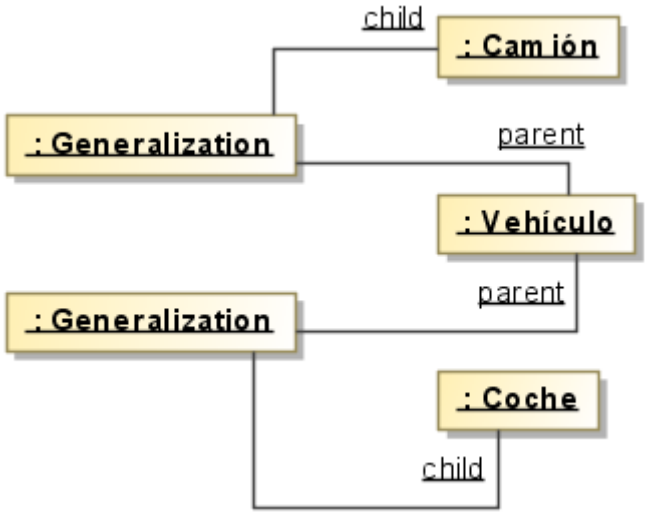


Figura 5.6 Instanciación de generalización del metamodelo de UML

5.1.4 Restricciones

En el subconjunto del metamodelo mostrado en la figura 5.7, se define el nexo que une el lenguaje UML con las expresiones de un lenguaje para especificar restricciones, como podría ser OCL.

La metaclase *Constraint*, representa la restricción y su contenido, la regla definida, se encuentra en el atributo *symbol* de la metaclase *Expression*. Por otro lado, se puede comprobar que existe una asociación entre las metaclases *Constraint* y *Element*. Ésta, indica a qué elemento o elementos se refiere la restricción.

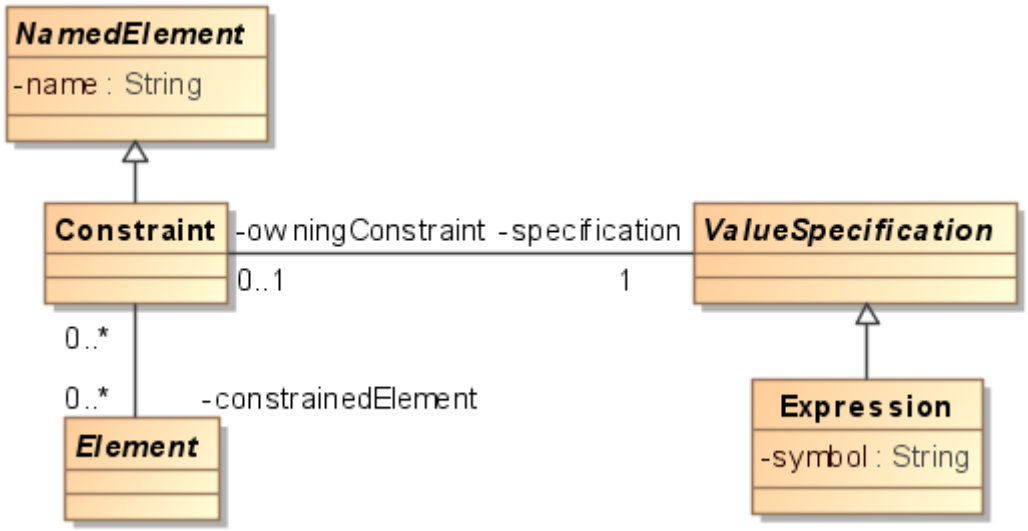


Figura 5.7 Restricción en el metamodelo de UML

5.1.5 Estereotipos

En la figura 5.8, se encuentra el fragmento del metamodelo relativo a la definición de estereotipos. Los estereotipos que se definen son instancias de la metaclase *Stereotype*, y sus atributos son instancias de la metaclase *Property*. Por otro lado, la asociación existente entre un estereotipo determinado y la metaclase que extiende, viene representada por la clase *Extension*.

Esa clase está relacionada con dos elementos. Uno de ellos es el estereotipo, mientras que el otro debe ser una metaclase definida en el metamodelo. Tal y como se puede observar, el estereotipo que participa en *Extension* es instancia de *ExtensionEnd*.

Por último, cabe destacar que en UML, los estereotipos que deseen aplicarse, una vez hayan sido definidos tal y como se ha explicado, deben estar contenidos en un *profile*.

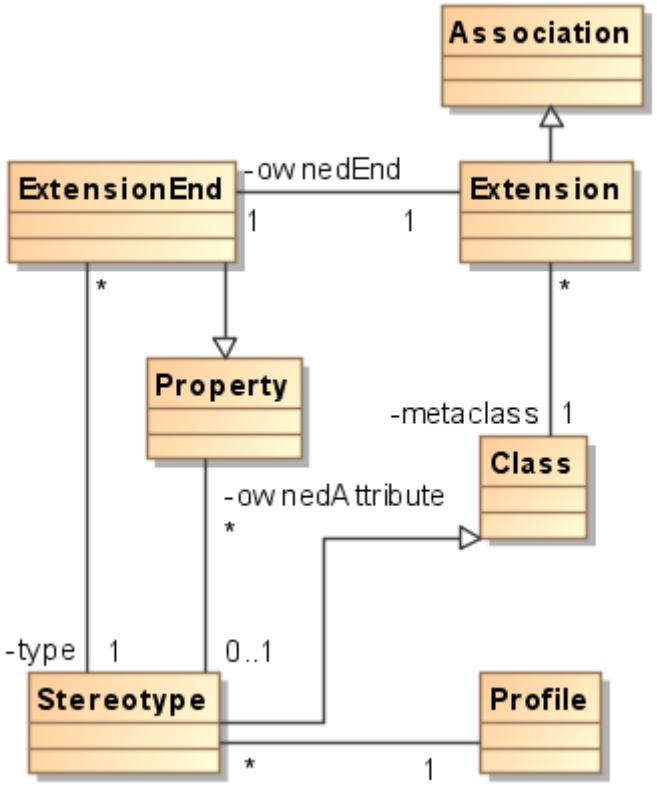


Figura 5.8 Estereotipo en el metamodelo de UML

En el ejemplo de la figura 5.9, se muestra la instanciación resultado de definir el estereotipo que se mostró en la figura 3.5. Entenderlo no resulta complicado, el atributo *name* de la instancia de la metaclase *Stereotype* toma por valor "*autoría*", que es el nombre del estereotipo en el ejemplo citado. Esta instancia, está relacionada con dos de la metaclase *Property*, puesto que el estereotipo contiene dos atributos, *autor* y *fecha*. Asimismo, esas dos instancias se relacionan con la de su tipo primitivo, que es *String*. El resto de clases y asociaciones, indican que el estereotipo sólo se puede aplicar a elementos que sean instancia de la metaclase *Class* del metamodelo de UML.

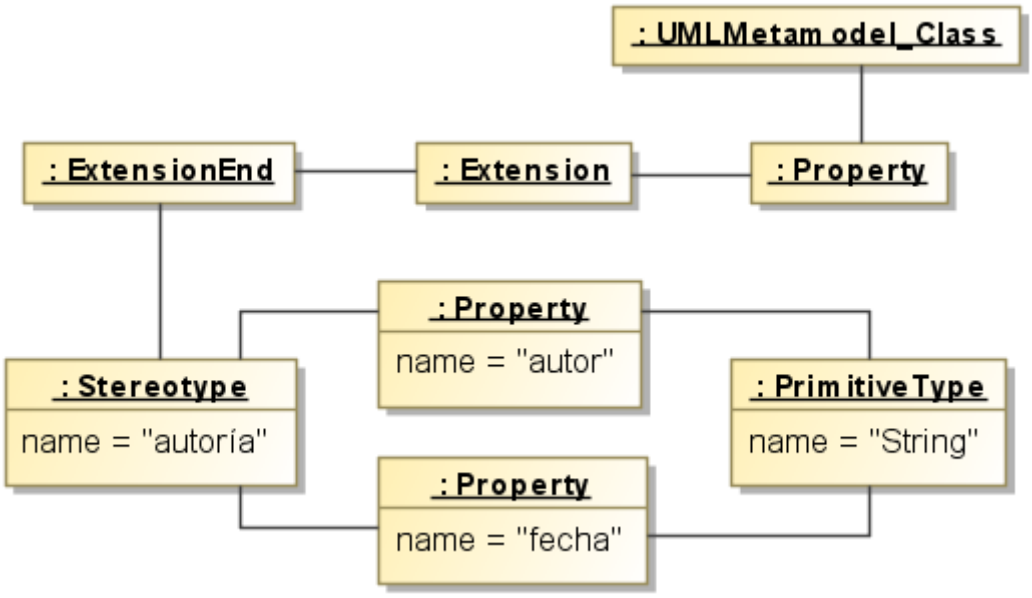


Figura 5.9 Instanciación de estereotipo del metamodelo de UML

5.1.6 Packages

En la figura 5.10, se encuentra el fragmento del metamodelo en el que se hallan definidas tanto la clase *Package* como sus asociaciones principales.

Los *packages* que se definen son instancias de la metaclase *Package* y todos los elementos que contiene, son instancias de la metaclase *PackageableElement*. De esta última, son subtipos muchas de las metaclases que se han ido explicando en esta sección, tales como *Class*, *Association*, *Extension* y *DataType*.

Por otra parte, el lenguaje UML permite que un *package* determinado importe elementos de otros *packages* externos. Por cada elemento importado, existe una instancia de la metaclase *ElementImport*, cuyos participantes son por un lado el propio *package* importador y por el otro, cualquier instancia de *PackageableElement*, que representa el elemento importado.

Asimismo, UML también permite que un *package*, importe otros *packages* externos de forma íntegra. Este hecho se instancia de una forma muy similar al expuesto anteriormente. Por cada *package* que se importa, existe una instancia de *PackageImport*, que se relaciona con el *package* que importa y el que es importado.

Además, se puede encontrar la metaclase *Model*, un subtipo de *Package*. *Model*, representa cualquier modelo que podamos definir para un sistema de información concreto.

Para finalizar, por cada *profile* que se aplique a un *package*, existe una instancia de la metaclase *ProfileApplication* que está relacionada con otras dos, la del *profile* aplicado y la del *package* que lo aplica.

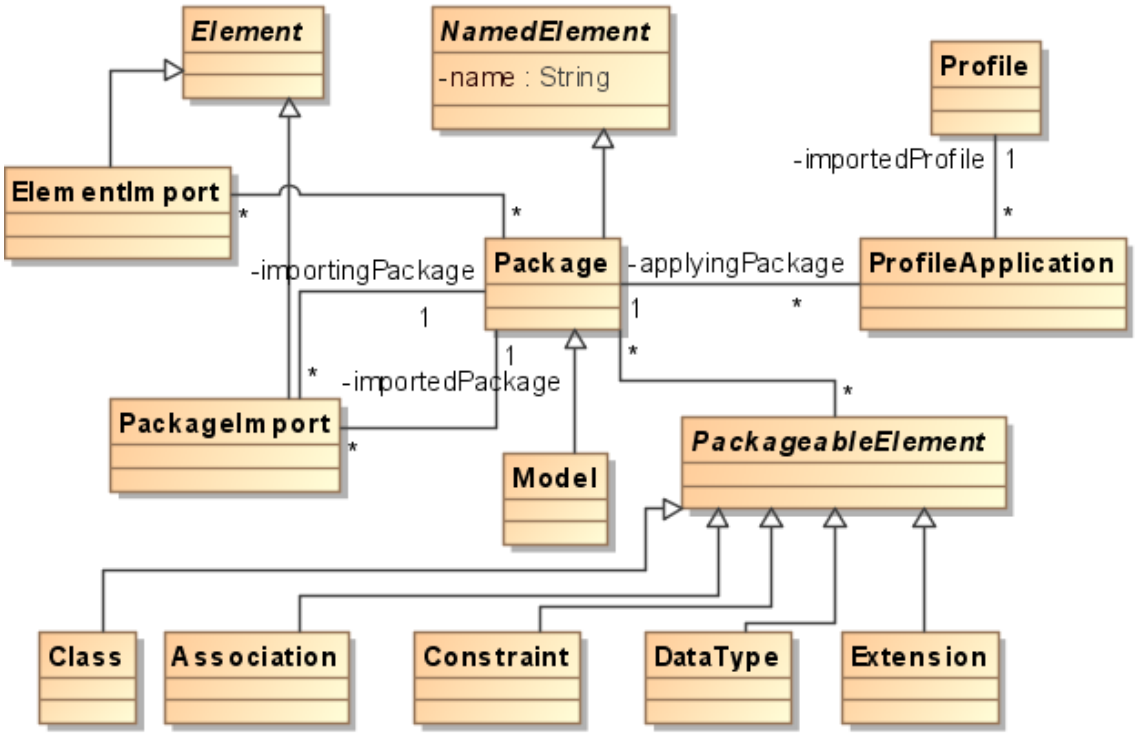


Figura 5.10 Package en el metamodelo de UML

6 XMI

En el ámbito de la informática, frecuentemente se desea que dos o más sistemas de información distintos puedan comunicarse e intercambiar información entre ellos. Cuando esto ocurre, los ingenieros del software deben decidir cuáles serán las entidades relevantes y sus relaciones para el ámbito en el que operan esos sistemas, esto es, deben definir esquemas conceptuales comunes para ellos. No obstante, este hecho por sí solo no es suficiente para garantizar esa comunicación, puesto que también resulta necesario un acuerdo en cuanto al formato de representación.

Imaginemos que los esquemas conceptuales se definen empleando el lenguaje UML. Una mera representación gráfica de dichos esquemas no constituiría una manera conveniente de realizar esas comunicaciones entre sistemas. Esto es debido a que no se han estandarizado las características gráficas que los esquemas UML deben seguir, y a que resulta mucho más sencillo procesar información en formato textual.

Con el objetivo de solventar esos problemas de comunicación, surge el *XML Metadata Interchange* (XMI), un estándar para representar esquemas e instancias en XML. Dicho estándar, al igual que UML, está respaldado por la OMG.

XMI es un estándar extenso y complejo, por ello, en este capítulo tan sólo nos centramos en proporcionar un ejemplo de esquema UML descrito mediante XMI, que será suficiente para poder entender posteriormente, los esquemas resultantes producidos por el transformador de modelos que se ha desarrollado. La especificación completa de XMI, puede encontrarse en [XmiEsp].

El esquema conceptual de la figura 6.1 será el que emplearemos para mostrar cómo representar un esquema UML en XML siguiendo la especificación de XMI.

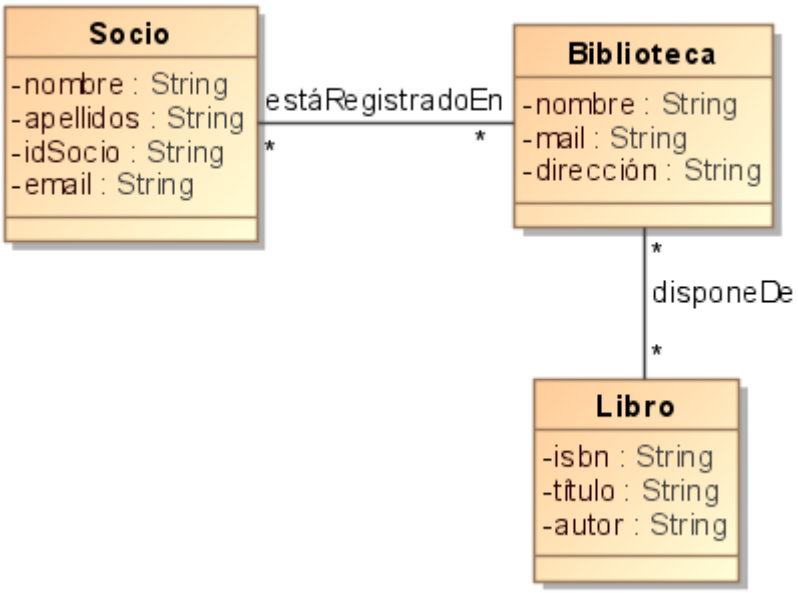


Figura 6.1 Fragmento esquema conceptual UML del concepto biblioteca

Para empezar, los ficheros XMI siguen la estructura general que puede apreciarse en el siguiente ejemplo.

```
<xmi:XMI xmi: version="2.1"
xmlns:xmi="http://schema.omg.org/spec/XMI/2.1>
  --Definición esquema UML--
</xmi:XMI>
```

Ejemplo 6.1 Estructura de un fichero XMI

Como se puede observar, los ficheros XMI se estructuran mediante etiquetas, todos los conceptos expuestos deben estar incluidos en etiquetas. Éstas son contenedores que se definen mediante un inicio, indicado por los caracteres "<" y ">", como podría ser *<xmi:XMI>*, y un final, indicado por medio de un concepto definido entre "</" y ">", como *</xmi:XMI>*.

Todo lo que se encuentre definido entre esos dos marcadores, se considera miembro de esa etiqueta. Así en el caso anteriormente expuesto, todo lo definido en el lugar indicado mediante *–Definición esquema UML--*, es miembro del concepto *xmi:XMI*.

Con el fin de poder explicar de manera más clara el contenido del fichero XMI del ejemplo, se ha optado por presentarlo de forma fragmentada. En primer lugar, se explica cómo se define una de las clases, seguidamente cómo se define una de las asociaciones y para terminar, se proporciona el fichero XMI completo.

En el ejemplo 6.2 se encuentra definida la representación XML de la clase *Biblioteca* del esquema proporcionado anteriormente.

```
<Class xmi:id = "C1" name = "Biblioteca" isAbstract = "false" >
  <ownedAttribute xmi:id = "A1" name = "nombre" lower=1 upper=1>
    <type xmi: type = "DT1" />
  </ownedAttribute>
  <ownedAttribute xmi:id = "A2" name = "mail" lower=1 upper=1>
    <type xmi: type = "DT1" />
  </ownedAttribute>
  <ownedAttribute xmi:id = "A3" name = "dirección" lower=1 upper=1>
    <type xmi: type = "DT1" />
  </ownedAttribute>
</Class>
```

Ejemplo 6.2 Representación XMI de la clase "Biblioteca"

Como se puede comprobar, toda la información relativa a la clase *Biblioteca*, se encuentra dentro de la etiqueta *Class*. En este caso particular, se encuentra su nombre, su identificador, un booleano que indica si es abstracta y toda una serie de elementos *ownedAttribute*, que tal y como su nombre indica, son los atributos de la clase. Cada uno de ellos contiene a su vez un nombre, un identificador, las multiplicidades y una referencia a su tipo de datos.

A continuación, en el ejemplo 6.3, se muestra la definición en XMI de la asociación "disponeDe" junto a las *properties* de sus dos extremos.

```
<Association xmi:id = "AS1" name = "disponeDe" memberEnd =
 "PR1 PR2" />
<Property xmi:id = "PR1" lower = 0 upper = * type = "C1"
 association = "AS1" />
<Property xmi:id = "PR2" lower = 0 upper = * type = "C2"
 association = "AS1" />
```

Ejemplo 6.3 Representación XMI de la asociación "disponeDe"

En el ejemplo, se puede observar que para las asociaciones, además de definir su nombre e identificador, se encuentran las referencias a las *properties* de sus extremos. A su vez, en dichas *properties* se encuentra una referencia a su tipo y a la asociación a la que pertenecen.

Una vez explicados estos dos fragmentos, resulta sencillo entender el fichero XMI al completo, que se muestra en el ejemplo 6.4.

```
<xmi:XMI xmi: version="2.1"
 xmlns:xmi="http://schema.omg.org/spec/XMI/2.1>
  <DataType xmi:id = "DT1" name = "String" />
    <Class xmi:id = "C1" name = "Biblioteca" isAbstract = "false" >
      <ownedAttribute xmi:id = "A1" name = "nombre" lower=1 upper=1>
        <type xmi: type = "DT1" />
      </ownedAttribute>
```

```
      <ownedAttribute xmi:id = "A2" name = "mail" lower=1 upper=1>
        <type xmi: type = "DT1" />
      </ownedAttribute>
      <ownedAttribute xmi:id = "A3" name = "dirección" lower=1
       upper=1>
        <type xmi: type = "DT1" />
      </ownedAttribute>
    </Class>
    <Class xmi:id = "C2" name = "Libro" isAbstract = "false" >
      <ownedAttribute xmi:id = "A4" name = "isbn" lower=1 upper=1>
        <type xmi: type = "DT1" />
      </ownedAttribute>
      <ownedAttribute xmi:id = "A5" name = "título" lower=1 upper=1>
        <type xmi: type = "DT1" />
      </ownedAttribute>
      <ownedAttribute xmi:id = "A6" name = "autor" lower=1 upper=1>
        <type xmi: type = "DT1" />
      </ownedAttribute>
    </Class>
    <Class xmi:id = "C3" name = "Socio" isAbstract = "false" >
      <ownedAttribute xmi:id = "A7" name = "nombre" lower=1 upper=1>
        <type xmi: type = "DT1" />
      </ownedAttribute>
      <ownedAttribute xmi:id = "A8" name = "apellidos" lower=1
       upper=1>
        <type xmi: type = "DT1" />
      </ownedAttribute>
      <ownedAttribute xmi:id = "A9" name = "idSocio" lower=1
       upper=1>
        <type xmi: type = "DT1" />
      </ownedAttribute>
      <ownedAttribute xmi:id = "A10" name = "email" lower=1 upper=1>
        <type xmi: type = "DT1" />
      </ownedAttribute>
    </Class>
    <Association xmi:id = "AS1" name = "disponeDe" memberEnd = "PR1
     PR2" />
    <Property xmi:id = "PR1" lower = 0 upper = * type = "C1"
     association = "AS1"  />
    <Property xmi:id = "PR2" lower = 0 upper = * type = "C2"
     association = "AS1" />
     <Association xmi:id = "AS2" name = "estáRegistradoEn"
     memberEnd = "PR3 PR4" />
    <Property xmi:id = "PR3" lower = 0 upper = * type = "C1"
     association = "AS2" />
    <Property xmi:id = "PR4" lower = 0 upper = * type = "C3"
     association = "AS2" />
</xmi:XMI>
```

Ejemplo 6.4 Representación XMI de un esquema relativo a bibliotecas

HL7

7 INTRODUCCIÓN AL ESTÁNDAR

Teniendo presente que el objetivo de este proyecto es transformar los modelos del estándar de salud HL7 a UML/OCL, resulta obligado presentar los elementos que intervienen en HL7 con un amplio nivel de detalle para posteriormente poder comprender en su totalidad los capítulos dedicados al desarrollo de la herramienta de transformación.

En el bloque que se inicia con este capítulo, se presenta una introducción al estándar HL7, se describen los modelos de información que lo componen y los distintos elementos que intervienen en estos. Todo ello, acompañado de ejemplos con el objetivo de conseguir que las explicaciones resulten lo más entendedoras posible.

7.1 Health Level Seven

Health Level Seven (HL7) es una organización sin ánimo de lucro, fundada en el año 1987 en los Estados Unidos y cuya misión, consiste en lograr una interoperabilidad real entre los distintos sistemas de información que dan soporte a las actividades de aquellas organizaciones pertenecientes al ámbito de la salud. A tal fin, promueve el desarrollo y la implantación de toda una serie de estándares, que se desarrollan por consenso en un entorno abierto y en los que participan actores tanto especialistas en el ámbito de las tecnologías de la información, como en el de la salud.

Actualmente, HL7 cuenta con unos 2.300 miembros y tiene presencia en más de 50 países, entre los que se encuentra España.

Cabe destacar también, que HL7 está acreditada por ANSI, asociación que se encarga de supervisar el desarrollo de estándares en los Estados Unidos, desde el año 1994.

Entre todos los estándares promovidos por HL7, el que goza de una mayor aceptación y nos ocupará en este proyecto, es el conocido como estándar HL7 V3, a partir de ahora, para simplificar, nos referiremos a él simplemente como estándar HL7. Este estándar permite que sistemas de información dispares puedan intercambiarse entre ellos datos clínicos y administrativos.

7.2 ¿Por qué Level Seven?

Level Seven, hace referencia al nivel siete del modelo de interconexión de sistemas abiertos, también llamado OSI (*Open System Interconnection*). Este modelo divide el conjunto de protocolos que forman parte de una red de ordenadores, en siete niveles independientes entre sí. En el nivel siete, también conocido como nivel de aplicación, tienen cabida todos aquellos protocolos de las aplicaciones que utilizan la red, por ejemplo *http* para aplicaciones web, *stmp* para aplicaciones de correo electrónico, etc.

7.3 La necesidad de un estándar en el ámbito de la salud

Por todos es conocido la enorme variedad de datos que se tratan en un sistema de información de un hospital cualquiera: información del personal médico y administrativo, de los pacientes, de las intervenciones realizadas, de los medicamentos suministrados, etc. Si se analiza la probabilidad que dos hospitales que tengan que tratar la misma información adopten un mismo modelo de datos, rápidamente se llegará a la conclusión de que ésta es irrisoria.

A continuación, se ilustra esta situación mediante un sencillo ejemplo.

Cada una de las clases que aparecen en la figura 7.1, podría ser la que representase el concepto paciente en dos sistemas de información distintos.

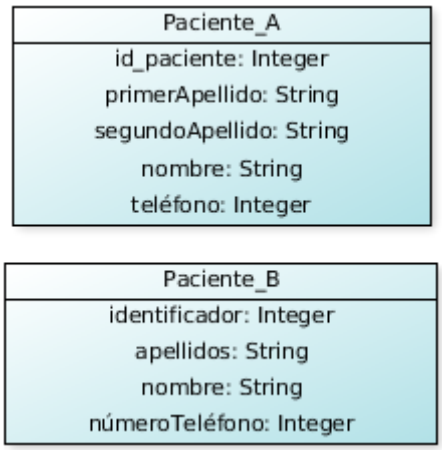


Figura 7.1 Concepto Paciente modelado de dos formas distintas

Se puede comprobar, que cada sistema utiliza una palabra distinta para hacer referencia a un mismo concepto, así por ejemplo, tanto *teléfono* en el primer sistema, como *númeroTeléfono*

en el segundo, se refieren al teléfono de contacto de un determinado paciente. Además, encontramos otras diferencias, en el primer ejemplo se utilizan dos atributos distintos para los apellidos, mientras que en el segundo se agrupan los dos en uno único.

En una situación como la ilustrada anteriormente, no sería posible trasladar el historial clínico de un determinado paciente de un sistema de información al otro de forma automática, debido a las diferencias existentes entre los modelos de información. Esta situación no es deseable en absoluto. No obstante, se podría solucionar con la ayuda de un componente de software que tuviese en cuenta las correspondencias que existen entre los dos sistemas.

Sin embargo, esta solución sólo solventa el problema a corto plazo. Si fuese necesario establecer una comunicación entre tres sistemas de información distintos, necesitaríamos tres componentes, si fuesen cuatro, serían seis los necesarios. Al llegar a los doce, ya serían sesenta y seis.

En la figura 7.2, en la que los vértices representan los sistemas de información y las aristas los componentes de software necesarios, se ejemplifica esta situación.

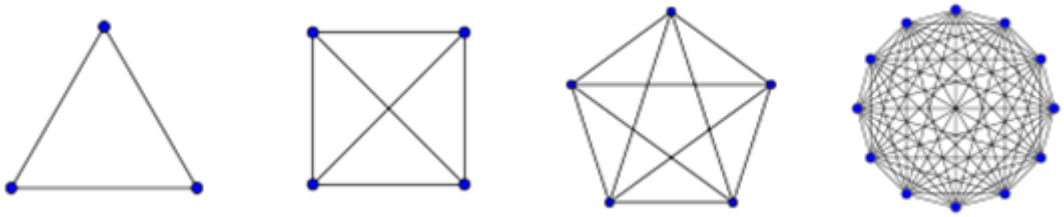


Figura 7.2 Componentes de software necesarios entre sistemas de información dispares

Analizando la problemática, se puede afirmar que el número de componentes de software necesarios es igual a N x (N – 1)/2, donde N representa el número de sistemas de información existentes. El número de componentes crece aproximadamente como la mitad del cuadrado del número de sistemas. Este argumento es suficientemente sólido por sí mismo como para plantearse una solución alternativa a la programación de componentes software adicionales.

En este escenario, HL7 propone una serie de estándares, con el objetivo que todas aquellas organizaciones que los adopten, no requieran de programación de componentes de software adicionales para que sus sistemas de información puedan comunicarse entre sí. A cambio, estas organizaciones deben comprometerse a cumplir con lo estipulado por los estándares, tarea que no resulta trivial.

7.4 Los ballots

El contenido del estándar HL7 no es decidido por unas pocas personas, sino que cada cierto tiempo se realizan una serie de votaciones para determinar qué debe formar parte de él y qué no, y en ellas, puede participar cualquier miembro afiliado a la organización.

Cada cuatro meses, más concretamente en enero, mayo y septiembre, se publica un nuevo *ballot* en la web del estándar. Un *ballot,* no es más que una página web, donde se puede encontrar toda la información necesaria para comprender el estándar y en la cual además, tal y como su nombre sugiere, se puede consultar todo el contenido que debe ser votado en ese ciclo (*ballot* significa votación en inglés). En [Hl7Ball] se pueden consultar todos los *ballots* publicados hasta la fecha.

A continuación, en la figura 7.3 se muestra una captura de pantalla del *ballot* de septiembre de 2010.

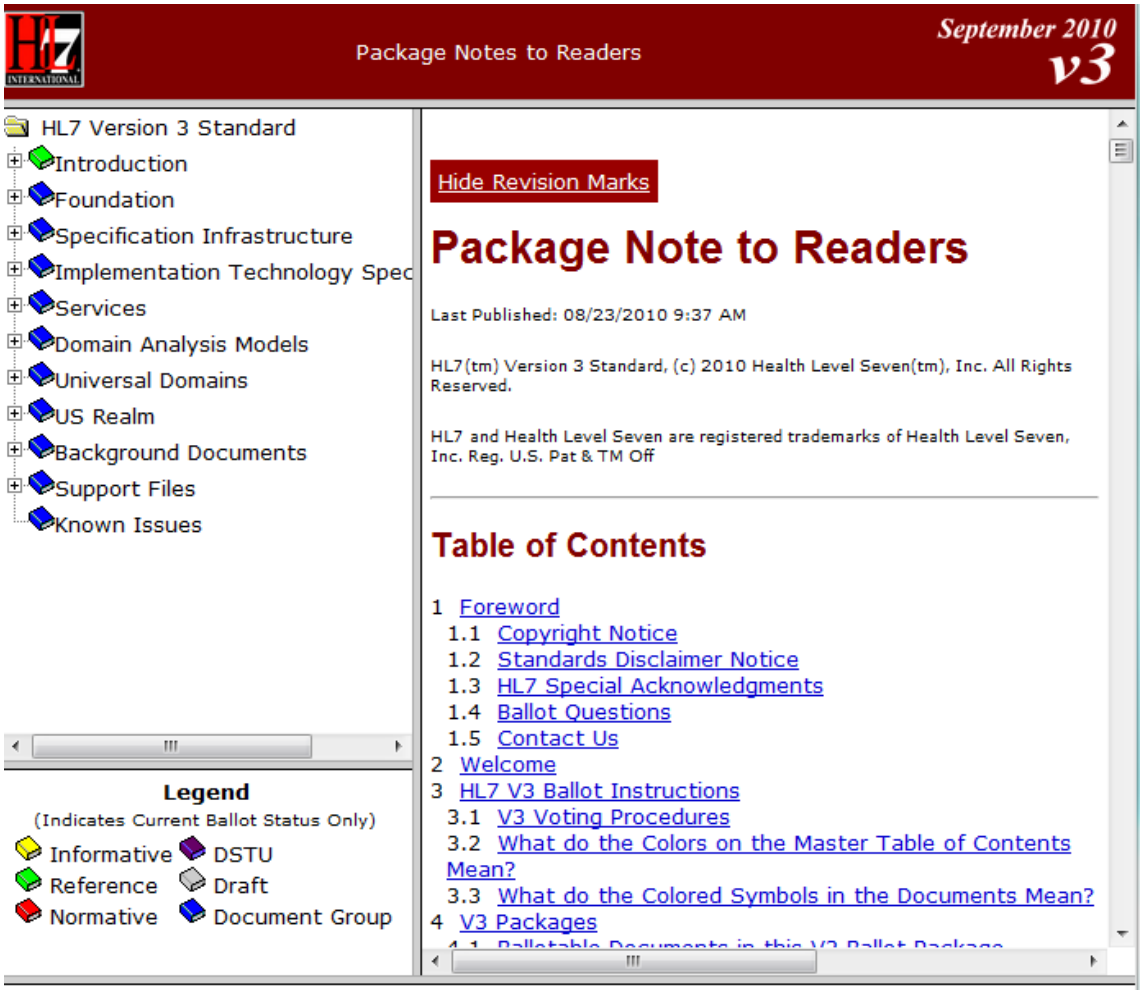


Figura 7.3 Captura de pantalla del ballot de HL7

Tal y como se puede observar, en la parte derecha se nos presenta el contenido del tema que se está consultando, mientras que en la izquierda, se nos muestran todas las secciones que componen ese *ballot*. Entre ellas, *Introduction* y *Foundation* resultan especialmente útiles, puesto que explican con detalle aquellos aspectos más relevantes del estándar. Por otro lado, la sección *Universal Domains,* presenta el contenido más importante para nuestro caso, puesto que en ella se pueden encontrar todos los modelos de información que componen el estándar.

En la figura 7.4, se puede apreciar que dichos modelos están agrupados en distintos dominios del ámbito de la salud, en total, el estándar distingue aproximadamente una treintena. En la captura, por temas de espacio sólo se muestra un subconjunto de ellos (desde facturación hasta registros médicos).

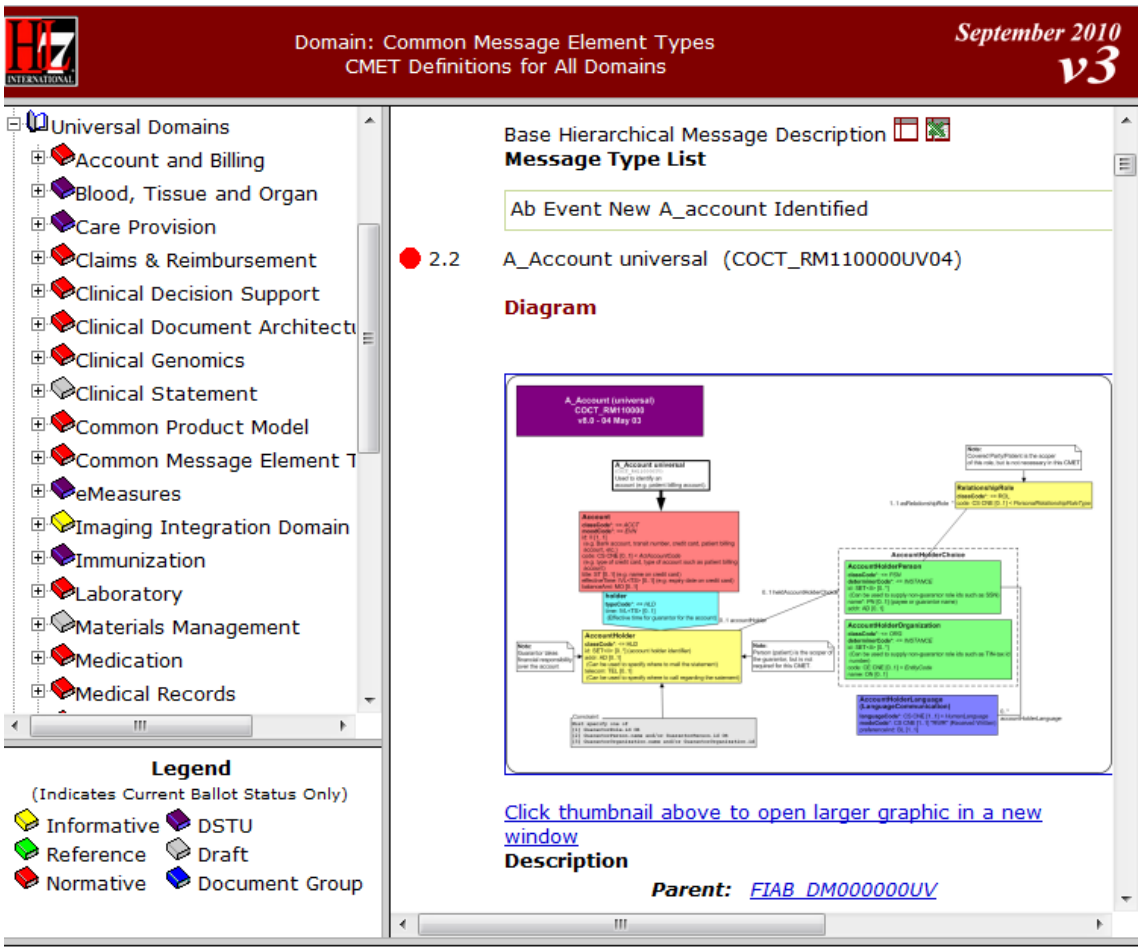


Figura 7.4 Dominios definidos en el estándar HL7

Además de resultar útiles para el público general de cara a poder consultar los detalles del estándar, los *ballots* también tienen por objetivo presentar de una forma clara el contenido que está sometido a votación a sus miembros.

Como se puede observar en las dos capturas de pantalla anteriores, cada tema o dominio está representado con un icono de distinto color. Estos colores indican la situación de ese tema concreto dentro del proceso de votación, tal y como indica la leyenda que hay en la esquina inferior izquierda de las capturas. En la tabla 7.1, se encuentra explicado el código de colores que se aplica.

Color	Significado
Amarillo	Documentos informativos. No son sometidos a ANSI para su aprobación y por este motivo, su método de votación es relativamente laxo.
Verde	Documentos de referencia. No son sometidos a votación, son los que explican las bases del estándar.
Rojo	Documentos normativos. Son sometidos a votación respetando rigurosamente los procedimientos descritos por ANSI, puesto que una vez son votados por la comunidad de HL7, son entregados a dicha organización para su aprobación.
Lila	Identifica documentos que han sido o serán aprobados para pruebas. Al final del periodo de pruebas pueden ser votados, modificados o descartados.
Gris	Borradores. Este tipo de documentos son meros borradores y no se pueden votar. Eventualmente, se podrán convertir en documentos informativos, de referencia o normativos.
Azul	Denota un grupo de documentos de cualquier tipo.

Tabla 7.1 Código de colores que aplica a los documentos de HL7

8 MODELOS DE INFORMACIÓN

En los *ballots* del estándar, se pueden encontrar varios tipos de modelos en los cuales se define la información que se acabará intercambiando entre distintos sistemas mediante el uso de mensajes XML. A continuación, se explican todos ellos y en una sección posterior, se analizarán con detalle los elementos que los componen.

***Reference Information Model* (RIM)**. Es el modelo de información que abarca todo el dominio del estándar. Los datos contenidos en todos aquellos mensajes que se acabarán intercambiando entre los distintos sistemas de información tienen como origen el RIM.

***Domain Message Information Model* (D-MIM)**. Es un refinamiento del RIM que incluye todos aquellos elementos necesarios para crear mensajes y documentos dentro de un área de interés particular, también conocidas como dominios. Actualmente, en el estándar existen más de quince dominios definidos entre los que se encuentran, por ejemplo, los de laboratorio, farmacia o gestión del personal.

***Refined Message Information Model* (R-MIM)**. Es un subconjunto de un D-MIM concreto, que incluye toda la información que resulta necesaria en un mensaje o conjunto de mensajes, adaptada para esos casos determinados.

***Hierarchical Message Descriptions* (HMD).** Se derivan de los R-MIM y contienen los campos exactos que un mensaje debe contener, además de indicar si son opcionales y su cardinalidad. Estos HMD, se representan mediante estructuras tabulares y se pueden obtener en el formato propio de *Microsoft Excel* o HTML.

Finalmente, a partir de los HMD, se obtiene el conjunto de reglas necesarias para construir un mensaje XML determinado, así como su contenido, que es el que se acabará intercambiando entre los distintos sistemas de información que adopten el estándar. Ese conjunto de reglas se define en ficheros XSD, contra los que se pueden validar los mensajes escritos en XML.

Nos centramos en los tres primeros tipos de modelos (RIM, D-MIM y R-MIM). Los HMD y los archivos XSD del estándar, no se explicarán con detalle, debido a que en el proyecto que nos ocupa, el interés radica en la transformación de los modelos de información, no se desea llegar al detalle del nivel de implementación de los mensajes que se intercambian.

Las reglas que deben aplicarse en la derivación de modelos más generales a otros más concretos, de los R-MIM a los HMD y de estos últimos a los ficheros XSD, se detallarán en un capítulo posterior, concretamente el 17, debido a que no sería posible comprenderlas en este

punto del documento, sin haber analizado antes los elementos que intervienen en los distintos modelos del estándar.

En la figura 8.1, extraída de la web de la página web del *ballot*, se representan las derivaciones de modelos explicadas.

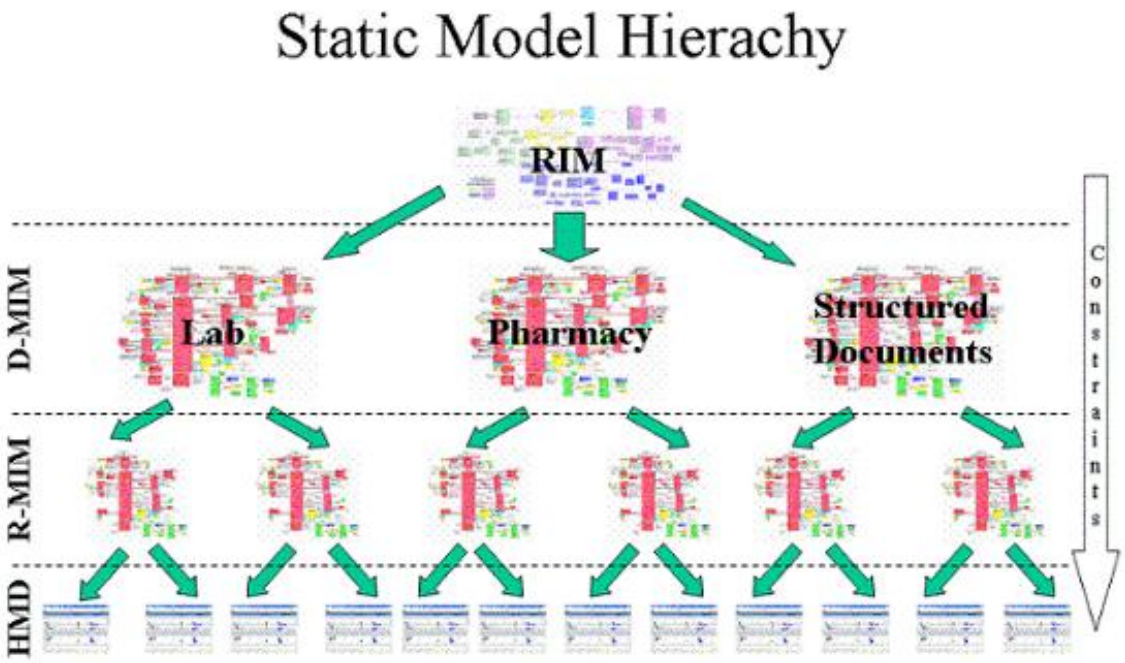


Figura 8.1 Jerarquía de los modelos de información de HL7, obtenida de [Hl7Ball]

8.1 RIM

EL RIM es el único de los modelos de información del estándar definido en UML. No obstante, sólo hace uso de un subconjunto muy reducido de todos los elementos que proporciona su especificación. Más concretamente, en él encontramos definidas clases, asociaciones, atributos y generalizaciones. Todos estos elementos se explicaron con detalle en el capítulo 3, dedicado a UML. Por este motivo, se procede directamente a explicar las particularidades de este modelo de información sin detenernos a analizar aquellas características que son comunes a cualquier modelo descrito empleando el lenguaje UML.

8.1.1 Clases

En el RIM se encuentran seis clases principales que constituyen los pilares fundamentales de los modelos de información del estándar. Cada una de estas clases define un gran número de

subclases, aunque sólo un pequeño subconjunto de ellas se encuentra representado de forma explícita en el RIM. Cabe destacar el nivel de abstracción al que se ha llegado en este modelo, puesto que hay más de 2.500 clases distintas definidas en el estándar y sin embargo, se ha conseguido definir el RIM de forma que sólo haya seis principales y todas las demás deriven de ellas.

Estas seis clases principales son: acto (*Act*), relación entre actos (*ActRelationship*), participación (*Participation*), rol (*Role*), relación entre roles (*RoleLink*) y entidad (*Entity*). De estas seis, *Act*, *Role* y *Entity* son las más importantes, puesto que las demás sirven para establecer asociaciones entre ellas.

Además, aparte de los mencionados, existe el tipo infraestructura donde tienen cabida todos aquellos conceptos relacionados con el lenguaje y el intercambio de mensajes. Este tipo, pese a aparecer en el RIM, no es considerado una de las clases principales.

A continuación, se explica qué representa cada una de las clases principales, y para los tipos *Act*, *Role* y *Entity*, que como se ha dicho anteriormente, son los más relevantes, se detallarán algunos de sus atributos y especializaciones más significativos. Definirlos todos queda fuera del alcance de este documento, puesto que sería una tarea demasiado laboriosa y además, no resulta necesario para que el lector pueda comprender los objetivos que se pretenden cubrir en este proyecto.

Por otro lado, para la mayoría de los casos se proporcionan ejemplos, pues en ocasiones las definiciones son demasiado abstractas y resultan necesarios para entender completamente los conceptos presentados.

En primer lugar, hay que tener presente que en el RIM y sus modelos derivados, se hace uso de un código que asigna a cada tipo de clase y sus subclases un color determinado. En la tabla 8.1, puede consultarse qué color tiene cada uno de los tipos de clases asignado.

TIPO DE CLASE	COLOR
Act	Rojo
ActRelationship	Rosado
Participation	Azul
Role	Amarillo
RoleLink	Amarillo claro
Entity	Verde

Infraestructure	Lila

Tabla 8.1 Código de colores que aplica a las clases de HL7

8.1.1.1 Acto

Es la representación de algo que se está haciendo, se ha hecho, se puede hacer o se ha solicitado que se haga. Posibles ejemplos de acto son una visita médica, una intervención quirúrgica o un diagnóstico.

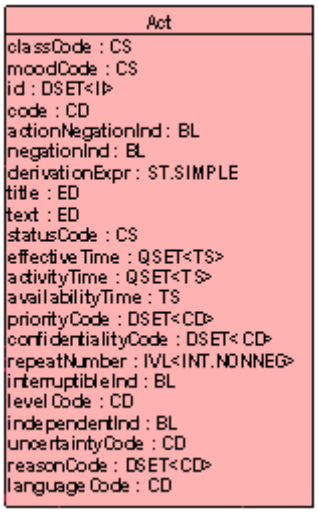


Figura 8.2 Clase Act del RIM, obtenida de [Hl7Ball]

Algunos de los atributos más importantes que presenta la clase acto son los siguientes:

- Classcode. Nos indica a qué tipo de acto pertenece la clase. A partir de la definición que se ha proporcionado anteriormente, se puede deducir que la cantidad de conceptos que pueden ser considerados actos es enorme, por ello se hace necesaria una clasificación por tipos.
- MoodCode. Determina si el acto ya ha ocurrido, es una petición para que ocurra, un objetivo o un criterio.
- StatusCode. Se refiere al estado del acto, puede tomar valores como nuevo, activo, cancelado, completado o interrumpido.
- ActivityTime y effectiveTime. El primero se refiere a la fecha en la que ocurre el acto mientras que el segundo, indica una fecha relevante para el mismo. Aclaremos esta diferencia con un ejemplo. Para un acto de prescripción médica, *activityTime* representaría la hora y fecha en la que el médico la escribió, mientras que

effectiveTime, haría referencia al tiempo durante el cual la medicación prescrita debe ser tomada.

Algunas de las subclases más importantes de *Act* son *Observation*, *Procedure*, *Supply* y *PatientEncounter*.

- Observation. Es un acto en el cual se obtiene algún tipo de información que es necesario anotar. Un ejemplo sería una revisión médica, ya que tras su finalización, es necesario guardar toda una serie de datos del paciente tales como la edad, la altura, el peso, etc.
- Procedure. Es un acto tras la finalización del cual, la condición física del sujeto que pasa por él cambia. De este modo, cualquier intervención quirúrgica sería considerada un *Procedure*.
- Supply. Se define como un acto que implica el aprovisionamiento de un determinado material por parte de una entidad a otra. Así, la venta de un medicamento sería un ejemplo de *Supply*.
- PatientEncounter. Es un acto que ocasiona una interacción entre un paciente y personal sanitario. Las visitas médicas, ya sean en un hospital o en el domicilio de un enfermo, serían un ejemplo de *PatientEcounter*.

8.1.1.2 Entidad

Representa seres vivos y cosas no vivas que existen o existirán en un futuro. También puede hacer referencia a un grupo de cosas. Ejemplos de entidades son:

- Seres vivos: personas, animales y plantas.
- Cosas no vivas: lugares y sustancias químicas.
- Entes abstractos: organizaciones tales como hospitales o centros de atención primaria.

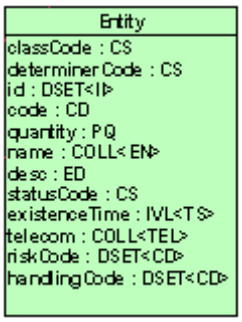


Figura 8.3 Clase Entity del RIM, obtenida de [Hl7Ball]

Sus atributos más relevantes son *classCode* y *determinerCode*.

- ClassCode. Indica qué tipo de entidad se representa.
- DeterminerCode. Se usa para distinguir entre instancias de entidades individuales, como una persona, y colectivos, que puede ser cualquier grupo de personas, por ejemplo.

Algunas de sus subclases más destacadas son ser vivo (*LivingSubject)*, cosa no viva (*Material)*, lugar (*Place)* y organización (*Organization)*.

8.1.1.3 Rol

Representa la responsabilidad o el papel que puede jugar una entidad. Los roles que se refieren a personas son normalmente puestos de trabajo o posiciones para los cuales están cualificadas, mientras que aquellos que se refieren a objetos inanimados, normalmente indican para qué se usan. Ejemplos de roles son:

- Referidos a personas: paciente, médico de cabecera y enfermero.
- Referidos a lugares: domicilio familiar y lugar de nacimiento.
- Referidos a papeles jugados dentro de organizaciones: empleado y proveedor.
- Referidos a cosas: medicamento y herramienta.

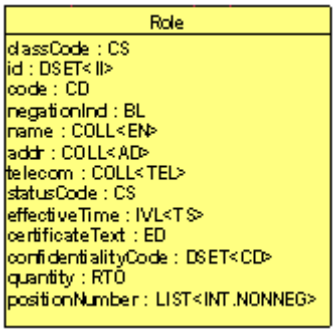


Figura 8.4 Clase Role del RIM, obtenida de [HI7Ball]

Una de las especializaciones más importantes que se pueden encontrar en la clase rol, tal y como puede esperarse teniendo en cuenta que se está hablando del ámbito de salud, es la de paciente.

8.1.1.4 Relación entre actos

Es una asociación dirigida que se establece entre dos actos. Uno de ellos juega el papel de origen (*source*) y otro de destino (*target*). Existen diversos tipos de relaciones entre actos, por ejemplo, puede ser que un determinado acto contenga, lleve a, actualice o cause otros actos. Así por ejemplo, podría existir una clase *ActRelationship* que relacionase un acto correspondiente a la extirpación de un tumor maligno con otro acto que representase la visita médica en la que ese tumor fue detectado. Otro ejemplo consistiría en tener un *ActRelationship* que relacionase un documento con su informe previo.

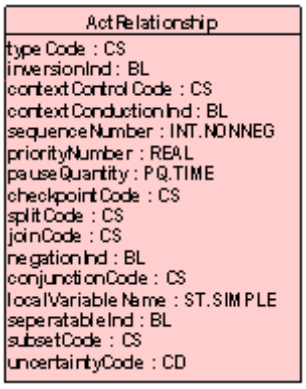


Figura 8.5 Clase ActRelationship del RIM, obtenida de [HI7Ball]

8.1.1.5 Participación

Una participación es una asociación que define el hecho que un rol se vea involucrado en un acto determinado. Muestra el comportamiento que una entidad, jugando un rol particular, tiene dentro del ámbito del acto al que se asocia la participación.

Una participación siempre va asociada a un acto determinado. En el momento que la actividad asociada a un acto acaba, la participación cesa del mismo modo.

Supongamos un caso en que una persona que juega el rol de cirujano, participa en un acto referente a una intervención quirúrgica como cirujano jefe o como asistente del cirujano jefe. Según la definición de participación dentro del estándar, tanto cirujano jefe como asistente del mismo, serían considerados participaciones en este caso particular.

Algunos ejemplos típicos de participaciones son los siguientes:

- Ejecutores del acto como cirujano, practicante, etc.
- Sujetos del acto. Lo más común dentro del ámbito de la salud, es que sea un paciente.
- Localización del acto.
- Otros como autor, receptor, emisor, beneficiario, etc.

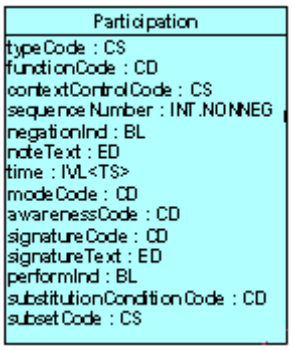


Figura 8.6 Clase Participation del RIM, obtenida de [HI7Ball]

8.1.1.6 Relación entre roles

Es una asociación que se establece entre dos roles determinados y al igual que ocurre en *ActRelationship*, existen los roles de *source* y *target*. Se utiliza, por ejemplo, cuando uno de los roles tiene autoridad sobre otro en una empresa organizada jerárquicamente, en las relaciones familiares y en las que se establecen entre los distintos integrantes de un mismo equipo médico. Así, podría existir un *RoleLink* que relacionase el rol de cirujano de una intervención

quirúrgica, con los roles de enfermero que también participan en dicha intervención. Cabe destacar, que este tipo de clase se encuentra en muy pocos modelos.

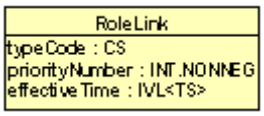


Figura 8.7 Clase RoleLink del RIM, obtenida de [Hl7Ball]

8.1.1.7 Infraestructura

Además de los seis tipos de clases expuestos, encontramos otro, las clases de tipo infraestructura, que hacen referencia al idioma en que se comunica una entidad y al intercambio de mensajes en general. Aunque en un principio se podría pensar que las instancias de esta clase sólo estarían relacionadas con entidades que fuesen personas, no es cierto, también pueden estarlo con entidades que sean máquinas capaces de transmitir mensajes de forma automática. En el RIM, la clase de este tipo que aparece, se denomina *LanguageCommunication*.

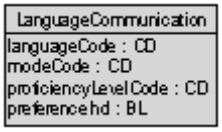


Figura 8.8 Clase LanguageCommunication del RIM, obtenida de [Hl7Ball]

8.1.2 Asociaciones

En la figura 8.9, se presenta un subconjunto del RIM. Se omiten las subclases que aparecen explícitamente y los atributos, con el objetivo de poder focalizar la atención del lector solamente en las relaciones que se establecen entre las distintas clases principales.

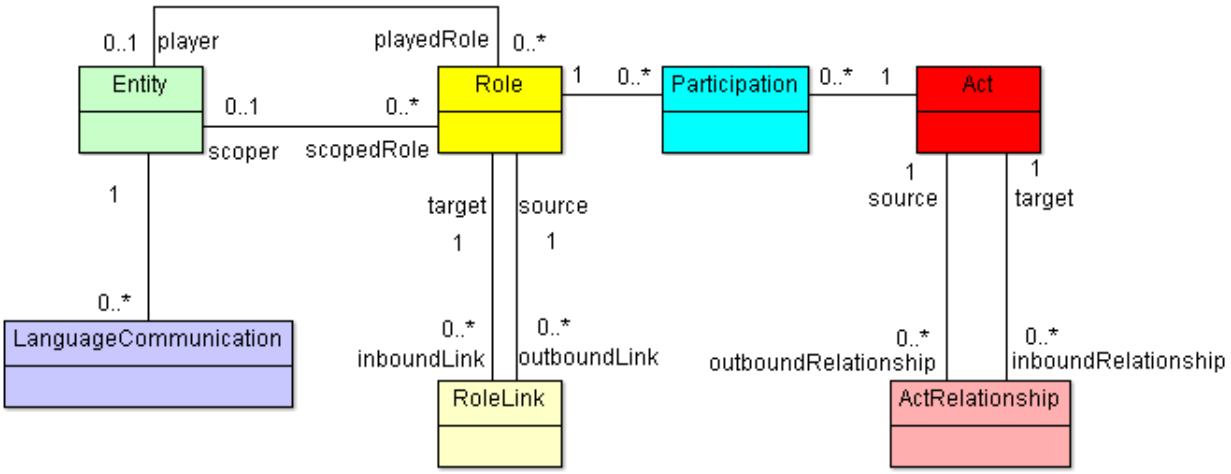


Figura 8.9 Asociaciones del RIM

De nuevo, cabe destacar el nivel de abstracción al que se llega en el RIM. Con tan sólo nueve asociaciones definidas en el modelo, se puede expresar cualquiera de las que se dan entre las más de 2.500 clases definidas en el estándar.

Las explicaciones proporcionadas anteriormente sobre el significado de cada una de las clases, son suficientes para entender las asociaciones del RIM. Sólo queda por aclarar que entre las clases Entidad y Rol, existen dos. Como se puede deducir a partir de los nombres de rol que aparecen en el esquema, en un caso, la entidad juega un rol determinado *(player*) y en el otro, define el ámbito en el que otra entidad juega ese rol (*scoper*). A continuación, se presenta un sencillo ejemplo para comprender esta distinción.

Supongamos que se desea representar el hecho que una persona es cirujana en un determinado hospital. Una instancia del RIM que expresa este hecho se muestra en la figura 8.10.

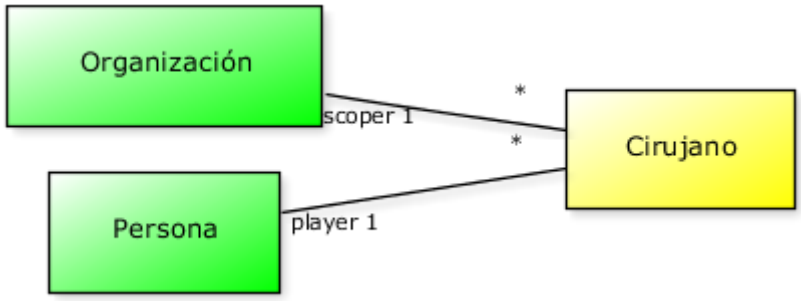


Figura 8.10 Instanciación del RIM (I)

Las clases personas y organización son entidades, y paciente es un rol, tal y como puede deducirse a partir del código de colores comentado anteriormente. La entidad que juega el rol (*player*) es persona y la que define el ámbito (*scoper*) es organización.

8.1.3 Ejemplo de instanciación

En la figura 8.11, se presenta un ejemplo más completo de instanciación del RIM, donde intervienen la mayoría de los tipos de clases anteriormente comentados.

En este caso, se modela el hecho que un médico de cabecera prescribe un medicamento a un paciente. El modelo se interpreta de la siguiente forma: una prescripción médica se asocia con una dosis prescrita, ya que son actos relacionados. Un médico juega el rol de médico de cabecera en un centro de atención primaria determinado y es el autor de la prescripción. Por su parte, una persona concreta juega el rol de paciente y habla una determinada lengua. Finalmente, aparece el farmacéutico, que juega el rol de vendedor y es el que recibe la prescripción.

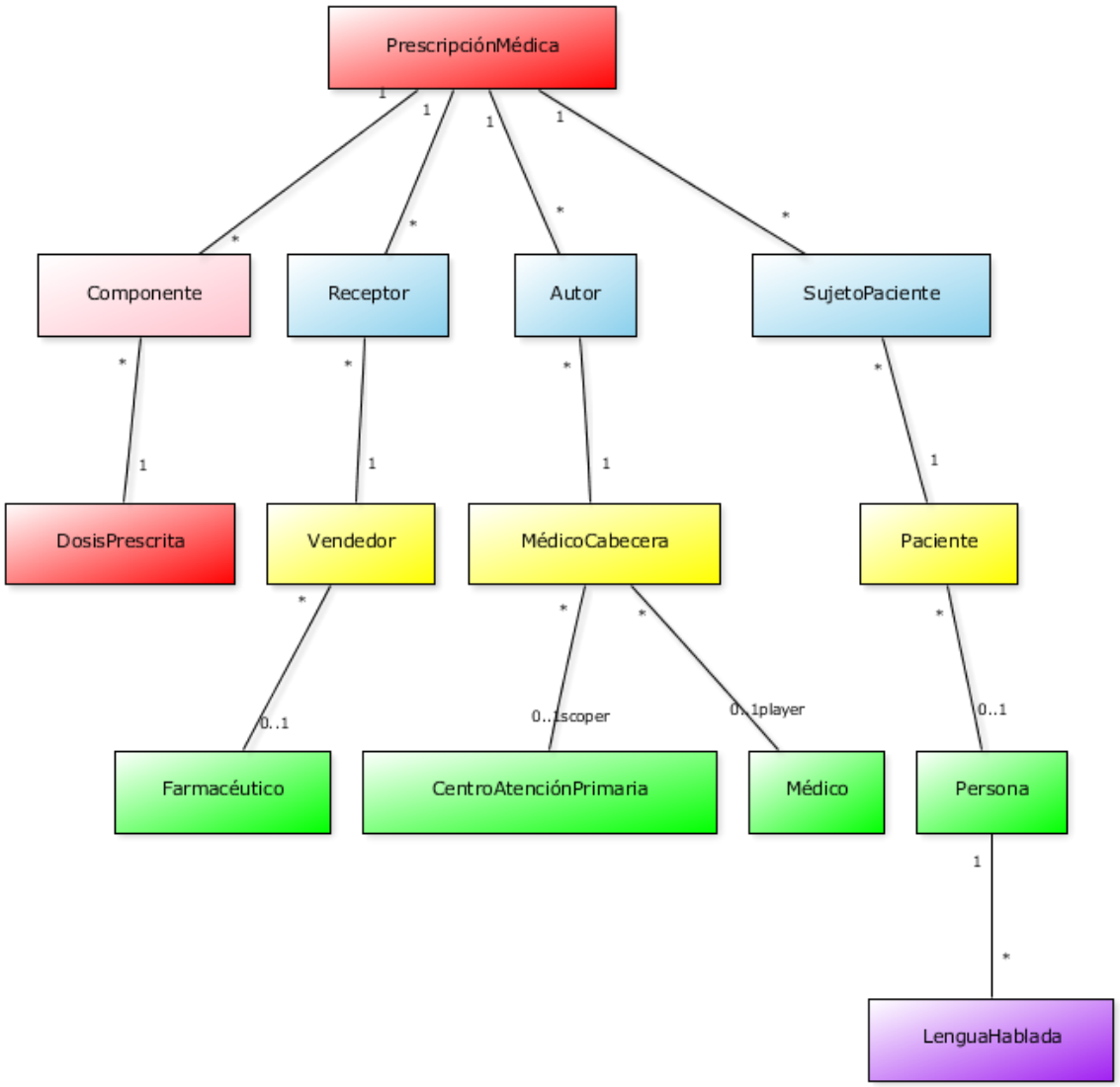


Figura 8.11 Instanciación del RIM (II)

8.1.4 Diagrama de clases del RIM al completo

Llegados a este punto, ya se han explicado todos los elementos que intervienen en el RIM. Para cerrar esta sección, se ofrece la versión completa del RIM que corresponde a la publicada en noviembre de 2010 (figura 8.12).

Por motivos de espacio, la imagen que se presenta en este documento es una versión reducida de la original. El lector que desee obtenerla con más definición puede acudir a [RIM10].

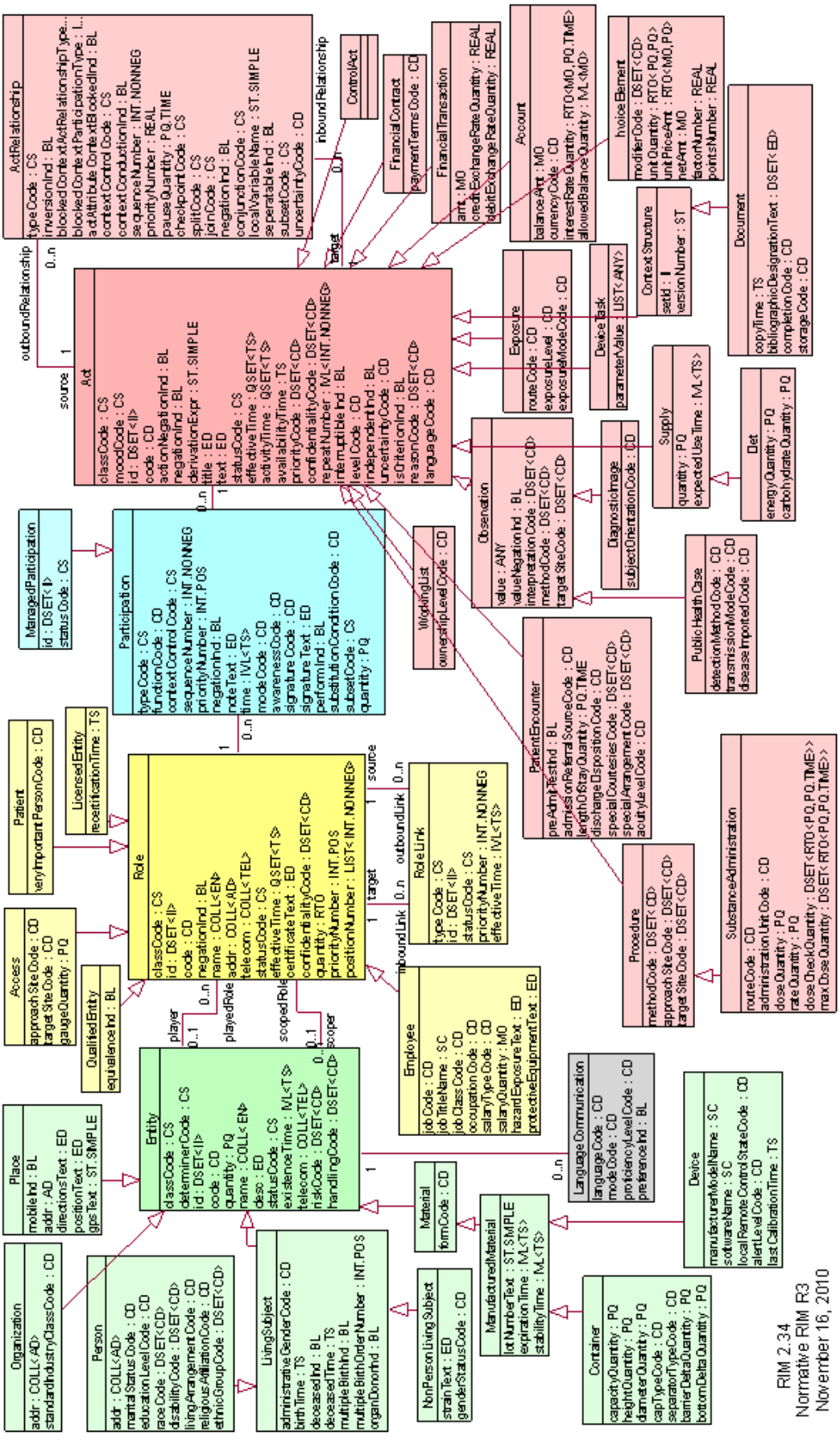


Figura 8.12 RIM, obtenido de [HI7Ball]

8.2 D-MIM y R-MIM

Los D-MIM y los R-MIM, a diferencia del RIM, no están descritos en UML, sino que están expresados en un lenguaje de modelización propio. La herramienta de diseño gráfico que se usa para elaborar estos diagramas es *Microsoft Visio*. Todos ellos se pueden encontrar bajo la sección *Universal Domains* de cualquier *ballot*.

Al no seguir ningún estándar ampliamente conocido, resulta necesario explicar con detalle los distintos tipos de elementos que podemos encontrar en este tipo de esquemas. Cada explicación va acompañada de uno o más modelos o fragmentos que han sido extraídos directamente del estándar.

8.2.1 Entry Point

Cada diagrama R-MIM cuenta con un *Entry Point*. A su vez, cada D-MIM consta de varios *Entry Points*, concretamente, como mínimo uno por cada R-MIM que derive de él. Se utiliza para señalar cuál es la clase más importante de ese esquema en concreto, también llamada clase focal, esto ayuda a distinguir cuál es el concepto principal. Lo más común, es que esta clase sea de tipo acto, aunque otro tipo de clases también pueden jugar este papel.

Los *Entry Points* se representan mediante rectángulos en los que podemos distinguir una flecha gruesa de color negro que apunta a la clase focal. Cada uno de ellos contiene un nombre, una cadena de caracteres que identifica inequívocamente a un D-MIM/R-MIM y una breve descripción.

En la figura 8.13, se muestra un ejemplo extraído del D-MIM del ámbito de programación de citas (*Scheduling*).

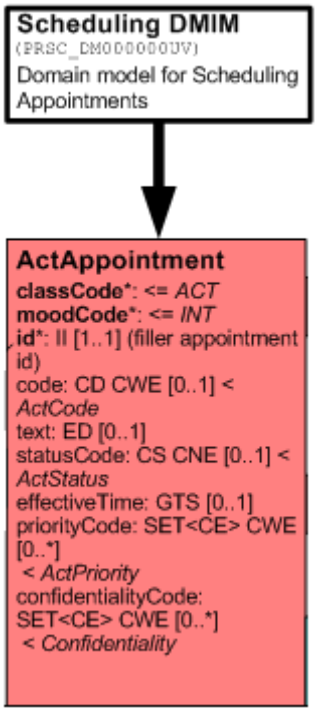


Figura 8.13 Entry Point de HL7, obtenido de [Hl7Ball]

En este ejemplo, *Scheduling DMIM* es el nombre del *Entry Point*, *PRSC_DM000000UV* el identificador del esquema y además, se encuentra una descripción que nos indica la información que contiene ese modelo concreto. Por otro lado, se puede observar que el acto *ActAppointment* constituye la clase focal del dominio.

8.2.2 Clases

Las clases de tipo *Entity*, *Role*, *Act* e *Infraestructure* aparecen representadas del mismo modo que se representen las clases en UML, es decir, mediante cajas.

A modo de ejemplo, en la figura 8.14 se muestran la entidad persona (*Person*) y el rol de miembro (*Member*).

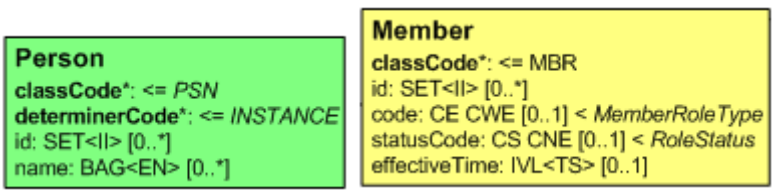


Figura 8.14 Clases de tipo Entity y Role, obtenidas de [Hl7Ball]

No ocurre lo mismo con las clases de tipo *ActRelationship*, *RoleLink* y *Participation*, que aparecen representadas en forma de flecha. En el caso de los tipos *ActRelationship* y *RoleLink*, la "cola" de la flecha está unida al elemento que juega el rol de *source* y la "punta" señala a la clase que actúa como *target*. Por otro lado, en el caso del tipo *Participation*, la "punta" de la flecha siempre señala a una clase de tipo *Role* y la "cola" está unida a una de tipo *Act*.

En la figura 8.15, se muestra una clase de tipo *ActRelationship* (*ScheduleRequest*), que relaciona dos actos entre sí. Concretamente, relaciona una cita (*ActAppointment*) con su petición (*ActAppointmentRequest*).

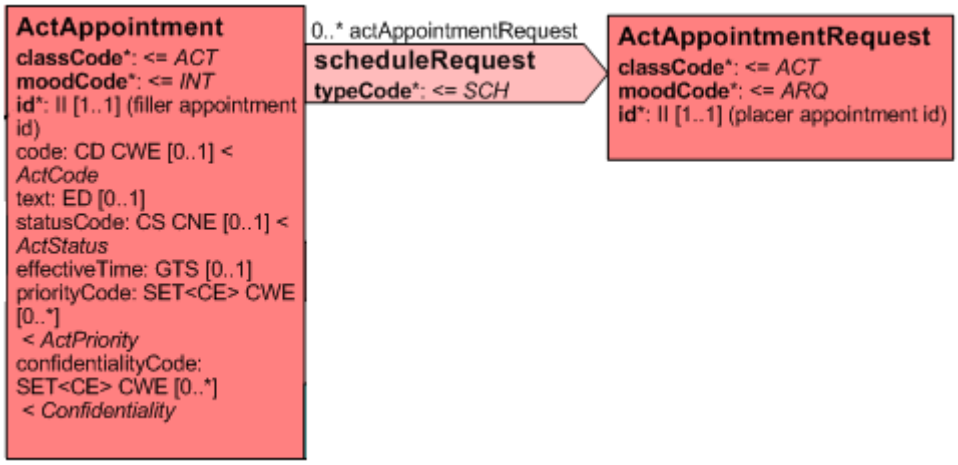


Figura 8.15 Clase de tipo ActRelationship, obtenida de [HI7Ball]

A continuación, en la figura 8.16, se muestra una clase de tipo *RoleLink* (*DirectAuthorityOver*) que establece una relación de autoridad de un patrocinador (*Sponsor*) sobre un asegurador (*Underwriter*).

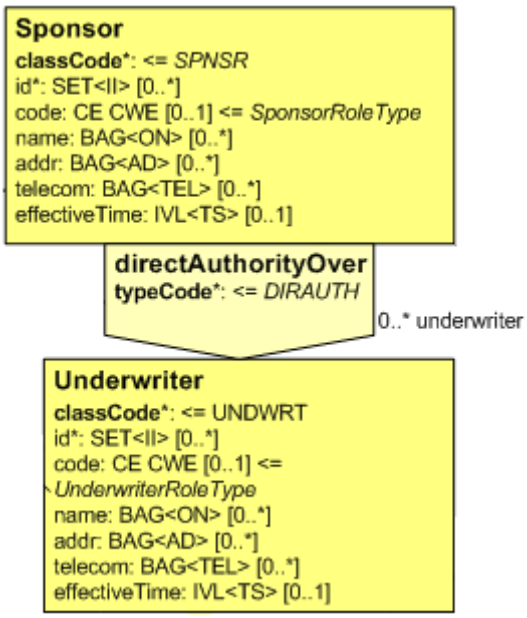


Figura 8.16 Clase de tipo RoleLink, obtenida de [Hl7Ball]

8.2.3 Asociaciones reflexivas

Tanto las clases de tipo *ActRelationship*, como las de tipo *RoleLink*, pueden representar asociaciones reflexivas, en tal caso, en lugar de ser representadas mediante una figura con forma de flecha, se denotan usando la tradicional caja, con la diferencia que en una de las esquinas encontramos una pequeña flecha.

En la figura 8.17, se ejemplifica la situación descrita. La clase *Precondition* de tipo *ActRelationship* es recursiva, puesto que indica que existe una asociación de precondición entre un criterio *(Criterion)* y otros criterios.

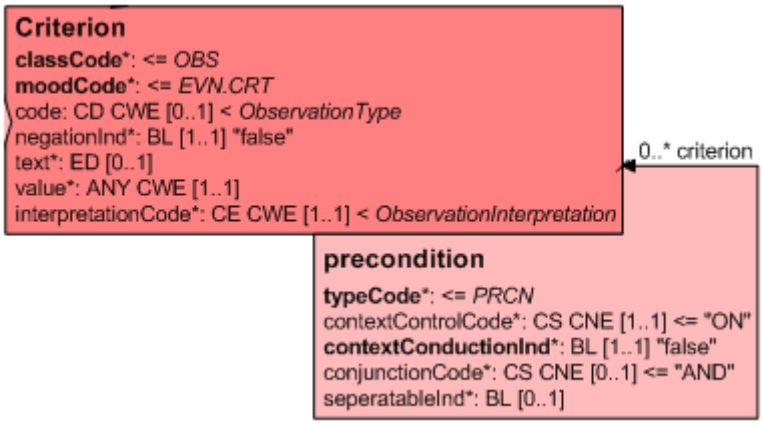


Figura 8.17 Asociaciones reflexivas en HL7, obtenidas de [Hl7Ball]

8.2.4 CMET

Los CMET (*Common Message Element Types*) son componentes predefinidos que se utilizan en varios modelos, cada uno de ellos se encuentra definido en un R-MIM propio. Desde el estándar se propone esta estructura con el objetivo de no repetir fragmentos que contengan la misma información en distintos modelos.

Se representan mediante cajas con un borde discontinuo y cada uno de ellos contiene un elemento principal, que corresponde a la clase a la que apunta el *Entry Point* del R-MIM que tiene asociado. El color de la caja que representa dicho CMET, es el mismo que el que corresponde a este elemento principal según la codificación anteriormente descrita.

Cada CMET contiene un nombre y un identificador, que es el mismo que el que tiene su R-MIM propio. Además, consta de un *Root Class Code*, que corresponde al valor del atributo *classCode* de su clase principal. Finalmente, presenta un nivel de atribución (*Attribution level*).

Así por ejemplo, cuando aparece la entidad organización, ésta siempre llevará asociada toda una serie de información fija: el nombre, la persona de contacto, si se encuentra asociada a otra organización, etc. Por ello, cada vez que esta entidad aparece, en lugar de representar toda esa información, con el objetivo de simplificar los modelos se representa una caja que nos indica en cuál podemos encontrarla.

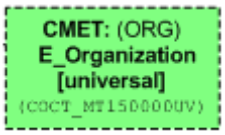


Figura 8.18 CMET de HL7, obtenido de [Hl7Ball]

En el ejemplo ilustrado en la figura 8.18, si se desease consultar toda la información asociada a la entidad organización, bastaría con acudir al modelo con identificador *COCT_MT150000UV*. Por otro lado, el nombre del CMET en este caso es *E_Organization*, su *Root Class Code* es ORG y su *Attribution level*, *universal*.

En la figura 8.19, se muestra el modelo al que se referencia en ese CMET. En él, se puede encontrar toda la información relativa a una organización. Concretamente, atributos como el nombre, la descripción y la dirección. Además, también se incluyen las personas de contacto y otras organizaciones con las que estaría relacionada en caso de pertenecer a un grupo de organizaciones.

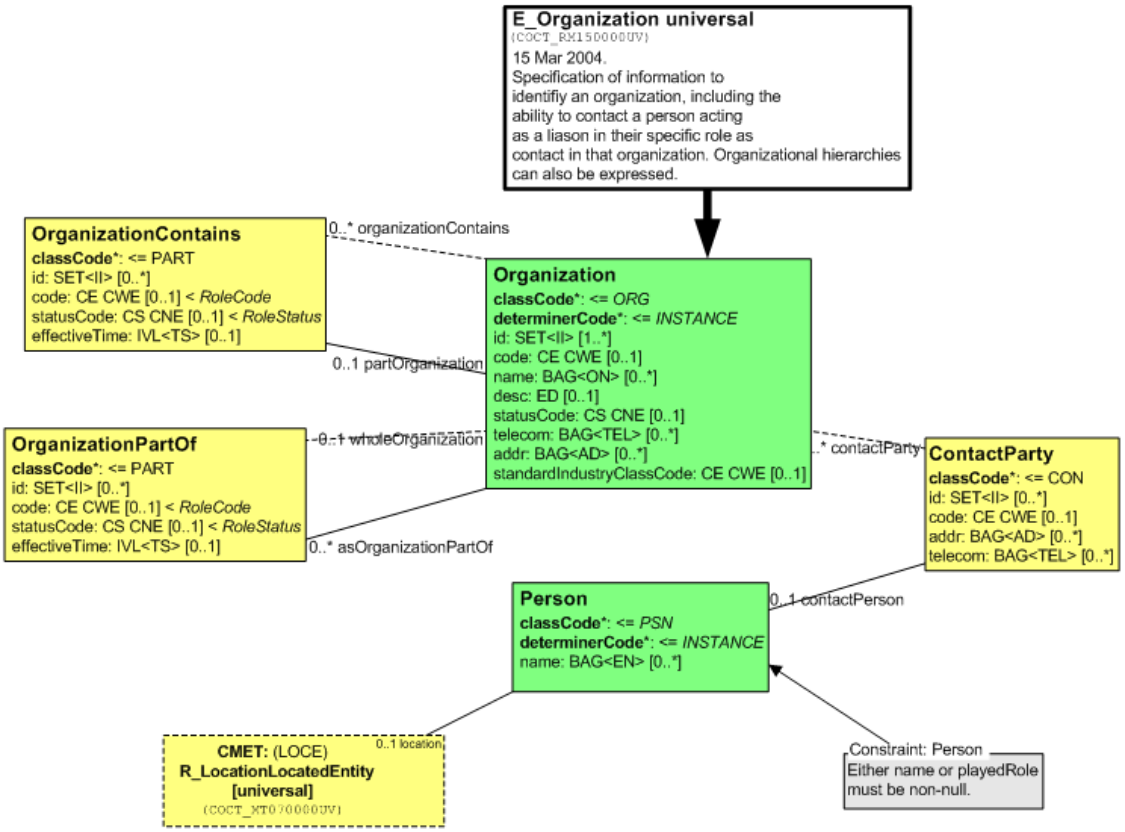


Figura 8.19 CMET expandido, obtenido de [Hl7Ball]

8.2.5 Choice

Los *choices* se utilizan para representar, de una forma intuitiva, el hecho que una asociación conectada a un grupo de clases determinado (*choice*), aplica a cualquiera de ellas. Es importante señalar, que todas las clases pertenecientes a un mismo *choice* deben ser del mismo tipo.

Un *choice* se representa mediante una caja con bordes discontinuos que en su interior contiene todas aquellas clases que forman parte de ese *choice*.

El que se presenta a modo de ejemplo en la figura 8.20, está extraído de un R-MIM perteneciente al dominio de los pagos y está relacionado con el rol de titular de la cuenta. Dicho titular, puede ser tanto una persona (*AccountHolderPerson*) como una organización (*AccountHolderOrganization*), por este motivo, se representan las dos entidades dentro de un mismo *choice* llamado *AccountHolderChoice*.

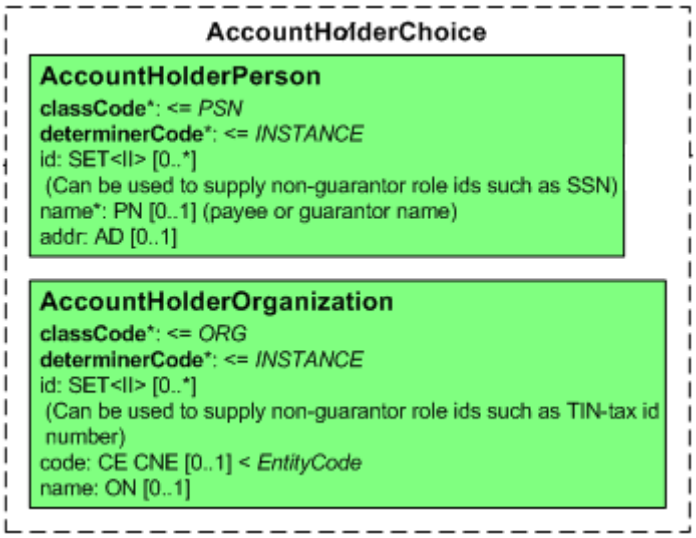


Figura 8.20 Choice de HL7, obtenido de [Hl7Ball]

El caso que se presenta en la figura 8.21, también extraído de un R-MIM del dominio de pagos, presenta algunas peculiaridades. En él, se puede comprobar que no es obligatorio que una determinada asociación deba aplicar a todos los miembros de un *choice* concreto.

Como se puede observar, el rol de pagador lo pueden jugar las entidades persona u organización. No obstante, carecería de sentido asociar la clase que representa el idioma que habla el pagador (*PayeeLanguage*) al *choice* completo, puesto que es lógico asociar el dominio de un idioma a una determinada persona, no a una organización.

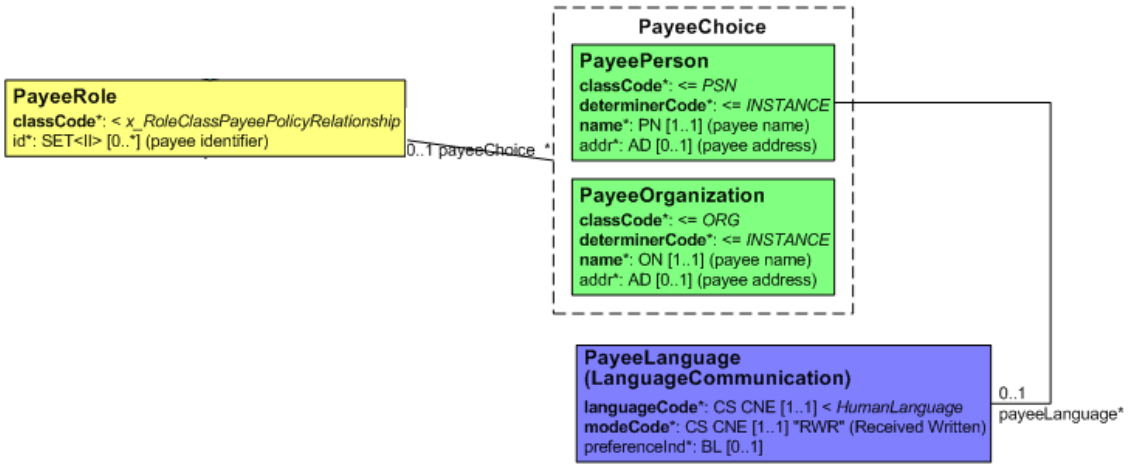


Figura 8.21 Asociaciones de un Choice, obtenidas de [Hl7Ball]

8.2.6 Nota

Las notas se utilizan para proporcionar información adicional así como aclaraciones, sobre una determinada clase, *choice* o CMET.

En el fragmento de modelo de la figura 8.22, se puede ver una nota referida a *Location*, una clase de tipo participación. La nota simplemente aclara que esa localización se refiere a la requerida para llevar a cabo el servicio o actividad que aparece en el mismo modelo.

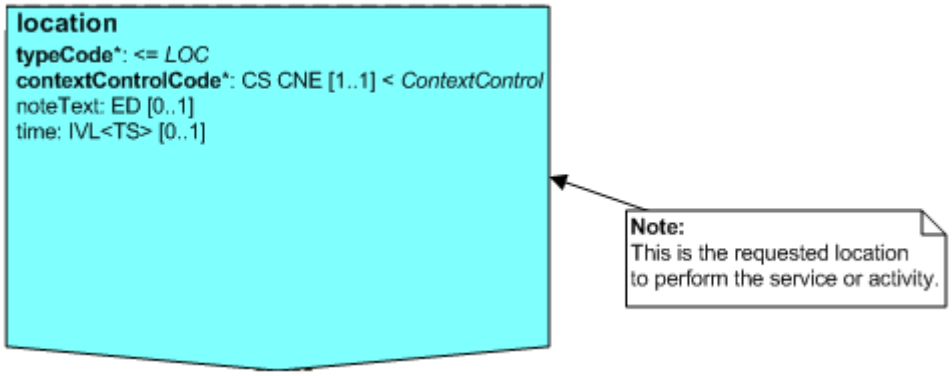


Figura 8.22 Nota de HL7, obtenida de [Hl7Ball]

Las notas como la que aparece en la figura 8.22 se denominan “notas de uso”. Además de éstas, existen notas de otro tipo que no aparecen representadas gráficamente en los esquemas, denominadas “notas descriptivas” y que contienen una breve descripción que puede hacer referencia tanto a clases como a atributos. En la gran mayoría de esquemas, existe una nota de este último tipo por cada clase y atributo que aparece y ese es el motivo por el que no aparecen representadas, puesto que con un número tan elevado de anotaciones, sería difícil distinguir lo que resulta realmente importante.

8.2.7 Restricciones

Las restricciones pueden afectar tanto a clases como a atributos concretos. Un ejemplo de restricción que podríamos encontrar para una clase podría ser el siguiente: “O bien el atributo X o bien el atributo Y, deben tener un valor diferente de nulo”. Las restricciones se representan mediante cajas que contienen la referencia a la clase o atributo que afectan y la restricción en sí, expresada en lenguaje natural.

En el ejemplo de la figura 8.23, se puede ver que se define una restricción sobre la clase dispositivo (*Device*). Dicha restricción indica que entre los atributos *id*, *code*, *manunfacturerModelName* y *softwareName* tiene que haber uno con valor distinto de nulo.

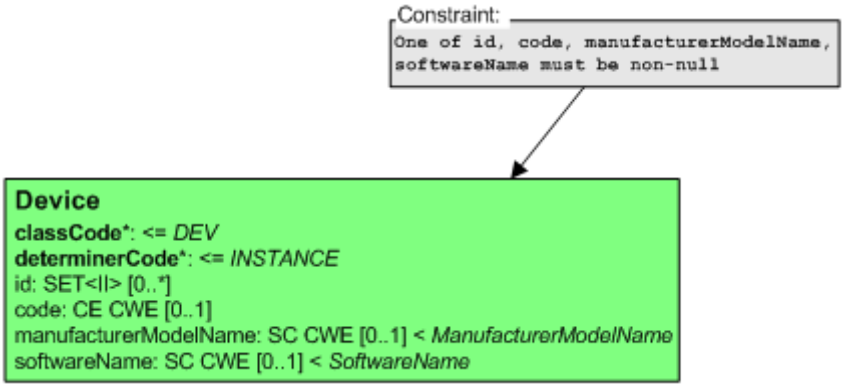


Figura 8.23 Restricción de HL7, obtenida de [Hl7Ball]

8.2.8 Clases repetidas

En algunos diagramas se pueden encontrar clases que no han sido coloreadas de forma uniforme, sino que aparecen rayadas, mostrando una mezcla de blanco y el color que les corresponde según su tipo. Este hecho, nos indica que esa misma clase aparece más de una vez en el modelo. Desde el estándar se propone esta técnica con el objetivo de no tener asociaciones que crucen el diagrama y que dificulten su comprensión.

A continuación, en el fragmento de diagrama representado en la figura 8.24 y extraído de un R-MIM del ámbito de documentos clínicos, se puede comprobar el uso de esta técnica.

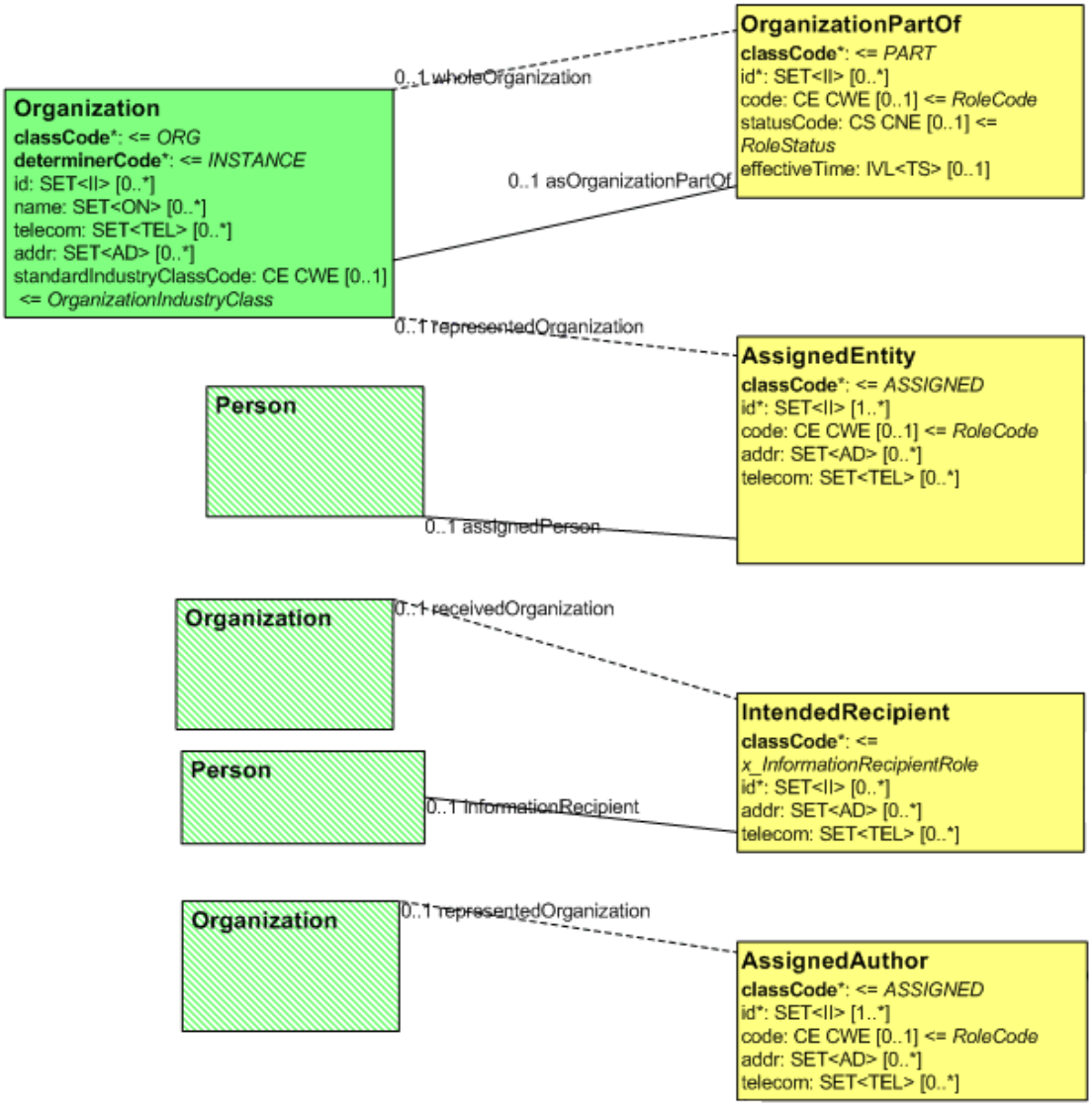


Figura 8.24 Clases repetidas en HL7, obtenidas de [Hl7Ball]

Como se puede apreciar, se cumple lo comentado anteriormente. Para no crear asociaciones que dificulten la lectura del modelo, la clase *Organization* aparece repetida y sólo en una ocasión aparece con un color de fondo uniforme. Por otro lado, la clase *Person* también aparece rayada porque se encuentra definida en otro punto del diagrama, que por cuestiones de espacio no se ha incluido en este documento.

8.2.9 Asociaciones scoper/player

Como ya se ha comentado anteriormente, entre las clases entidad y rol pueden darse dos tipos de asociaciones, en una la entidad juega el rol con el que se relaciona (*player*) y en la otra, representa el ámbito donde se juega ese rol (*scoper*).

Tanto en los D-MIM como en los R-MIM, para diferenciar entre los dos tipos de relaciones, las asociaciones donde el rol de la entidad es *player,* se denotan usando una línea continua, por el contrario, aquellas donde es *scoper*, se representan mediante una línea discontinua.

Para ilustrar esta característica del estándar, se utiliza el R-MIM con identificador *COCT_RM050100* completo (figura 8.25). Este modelo, contiene la mínima información necesaria para identificar a un paciente. Tal y como se puede observar, en este caso, *Person* juega el rol de paciente (*patientPerson*) y *E_Organization*, que es un CMET, define el ámbito donde la persona juega ese rol (*providerOrganization*).

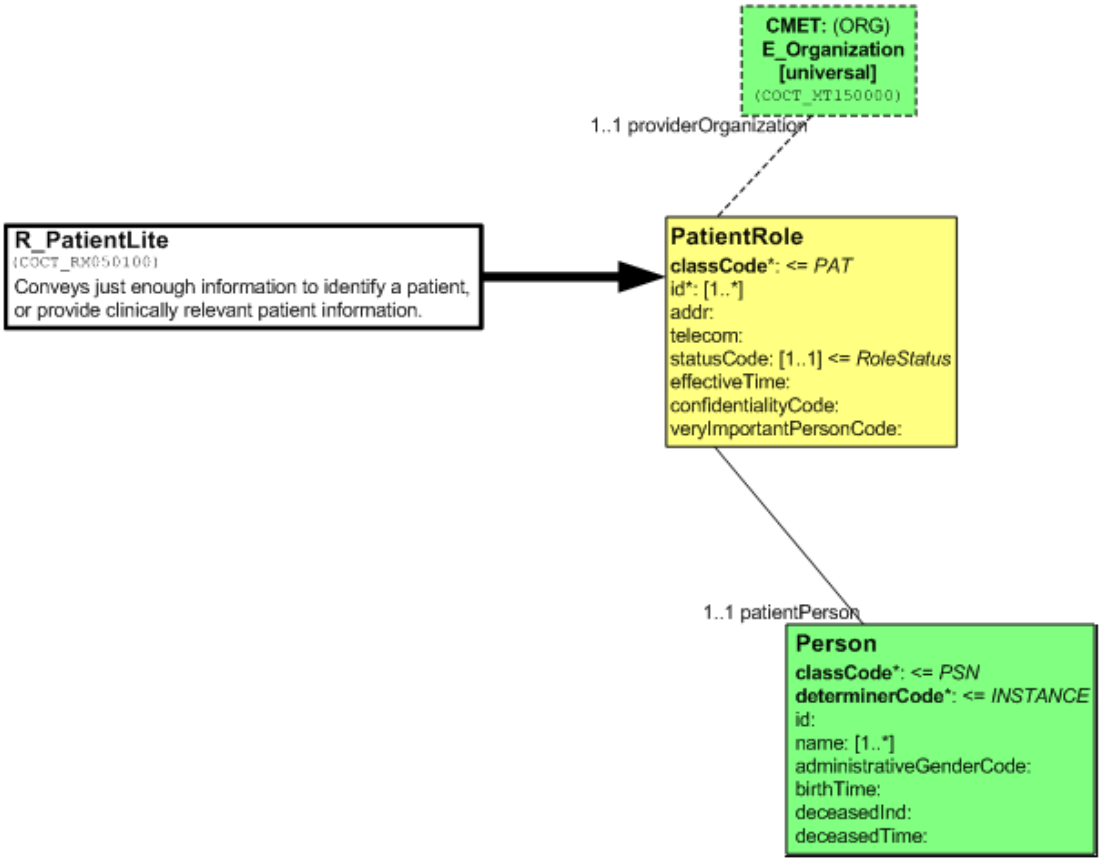


Figura 8.25 Asociaciones scoper/player de HL7, obtenidas de [Hl7Ball]

8.2.10 Atributos

En el capítulo 3, dedicado a UML, se explicó la forma de definir atributos en dicho lenguaje, que básicamente consiste en indicar el nombre de un atributo determinado y su tipo de datos separados por el carácter ':' acompañados de la multiplicidad entre corchetes. En HL7, la forma de definirlos no es tan sencilla, puesto que intervienen una amplia gama de situaciones a tener en cuenta:

- El atributo puede aparecer en negrita. Esto significa que siempre debe aparecer en los mensajes que se intercambian y que debe tener un valor distinto de nulo.
- El atributo puede aparecer con el carácter '*' a su lado. Esto indica que el atributo debe aparecer en los mensajes, pero que a diferencia de lo que ocurre en el caso anterior, puede valer nulo en caso de que se desconozca su valor. En caso que el atributo aparezca en negrita acompañado por el carácter '*', significa que es obligatorio y que debe tener un valor distinto de nulo.
- La multiplicidad del atributo se denota mediante corchetes siguiendo la misma notación que en UML. Así por ejemplo, [1..*] significa que dicho atributo debe tener como mínimo un valor y como máximo, un número cualquiera.
- El tipo de dato del atributo, aparece detrás del nombre del mismo, separado por el carácter ':'. Los tipos de datos definidos por el estándar HL7 serán presentados en un capítulo posterior.
- Si el tipo de dato es un código, va acompañado de las siglas CWE (*Coded with exceptions*) o CNE (*Coded no exceptions*). En el primer caso, se indica que al no existir un código definido para el concepto que se está representando, se puede optar por uno arbitrario, mientras que en el segundo, se indica que el código tiene que ser obligatoriamente uno de los definidos en el estándar.
- "<=" y "=" indican el dominio de vocabulario al cual debe pertenecer el valor de ese atributo. Un dominio de vocabulario no es más que un conjunto de conceptos que pueden ser considerados valores válidos para unos atributos concretos.
- Una cadena de caracteres entre comillas indica el valor por defecto para ese atributo.
- Para finalizar, puede incluirse una breve descripción del atributo paréntesis.

En la clase que aparece en la figura 8.26, se pueden observar los aspectos anteriormente descritos.

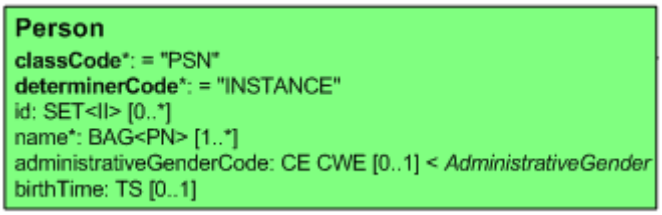


Figura 8.26 Clase Person de HL7, obtenida de [HI7Ball]

- En primer lugar, vemos que los atributos *classCode* y *determinerCode* son obligatorios, puesto que están en negrita, además, sus valores por defecto son "*PSN*" e "*INSTANCE*" respectivamente.
- *Id* es del tipo SET<II> (conjunto de identificadores de instancia) y tiene multiplicidad 0..*.
- *Name* es del tipo BAG<PN> (bolsa de nombres de persona) con multiplicidad 1..*, además, vemos que puede tomar el valor nulo por el carácter '*' que acompaña al nombre.
- *AdministrativeGenderCode* es un atributo de tipo CE (código) y vemos que puede tener un valor arbitrario ya que aparecen las siglas CWE. Además, se puede comprobar que la multiplicidad es 0..1 y que su valor debe pertenecer al vocabulario del dominio *AdministrativeGender*.
- Por último, *birthTime* es del tipo TS (punto temporal) y presenta una multiplicidad de 0..1.

La única característica que no cumple ese ejemplo, es la de contener comentarios entre paréntesis. En el ejemplo mostrado en la figura 8.27, se puede apreciar su uso.

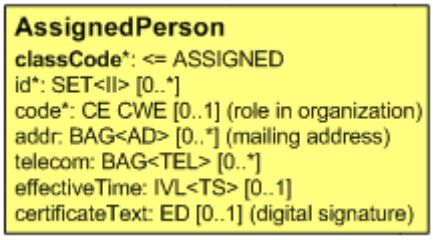


Figura 8.27 Clase AssignedPerson de HL7, obtenida de [HI7Ball]

Concretamente, se puede distinguir como en los atributos *code*, *addr* y *certificateText*, se han realizado algunas aclaraciones.

9 TIPOS DE DATOS

El estándar HL7, a diferencia de UML y OCL, contiene un gran número de tipos de datos distintos organizados siguiendo una estructura jerárquica. Estos tipos de datos, se han definido utilizando UML.

Uno de los principales inconvenientes que presentan los tipos de datos, es que la especificación UML proporcionada en los *ballots*, no coincide con la implementación XML de los mismos que se acaba usando en el intercambio de mensajes. Hay clases y atributos que no aparecen en la especificación que sí que aparecen en los ficheros XML, pero también se da el mismo caso en sentido contrario. Esto nos supone un inconveniente de cara al desarrollo de la herramienta de conversión y nos obliga a tomar toda una serie de decisiones que serán comentadas con detalle en la sección 18.1 del capítulo dedicado a los problemas del estándar HL7.

En este capítulo, nos limitaremos a proporcionar las descripciones de aquellos tipos de datos que son más relevantes y que están definidos tanto a nivel de especificación como de implementación.

9.1 Tipos básicos

En la tabla 9.1, se describen los tipos básicos definidos en el estándar HL7.

Tipo	Descripción
DataType	Es la clase de UML que representa un tipo de datos y corresponde al nivel más alto de la jerarquía de tipos de HL7.
ANY (*Data Value*)	Tipo abstracto del cual derivan los demás.
BL (*Boolean*)	Tipo booleano clásico, puede tomar los valores verdadero, falso y nulo.
BN (*Boolean Non Null*)	Tipo booleano que no puede valer nulo.

Tabla 9.1 Tipos de datos básicos de HL7

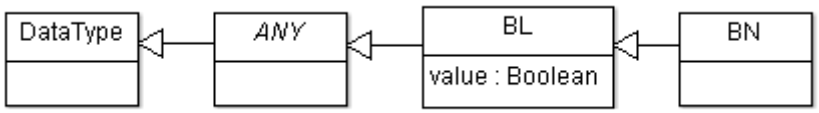


Figura 9.1 Diagrama de los tipos de datos básicos de HL7

9.2 Tipos de texto/multimedia

En la tabla 9.2, se muestran los tipos de datos de texto y multimedia.

Tipo	Descripción
BIN (*Binary Data*)	Datos binarios.
ED (*Encapsulated Data*)	Datos que pueden ser interpretados por personas o procesados por máquinas fuera del ámbito de HL7.
ST (*Character String*)	Cadena de caracteres, corresponde al tipo String de UML.

Tabla 9.2 Tipos de datos de texto/multimedia de HL7

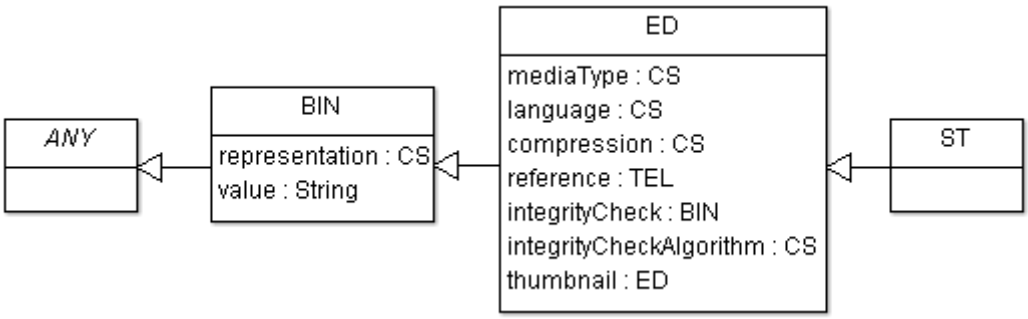


Figura 9.2 Diagrama de los tipos multimedia de HL7

9.3 Códigos

En la tabla 9.3, se pueden encontrar los tipos de datos que representan códigos.

Tipo	Descripción
CR (*Concept Role*)	Tipo de código que clasifica conceptos.
CD (*Concept Descriptor*)	Es un código que puede representar cualquier concepto, normalmente a través de un sistema de

	códigos previamente definido.
CS (*Coded Simple Value)*	Representa un código sin más.
CE (*Coded With Equivalents)*	Código que va acompañado de códigos de otros sistemas de códigos distintos al suyo que representan el mismo concepto.
CV (*Coded Value)*	Tipo en el que se especifica el código y el sistema de códigos en que éste es válido.
CO (*Coded Ordinal)*	Código en el que el sistema de códigos del que procede tiene un orden definido.
SC (*Character String with Code*)	Cadena de caracteres que puede ir acompañada de un código.

Tabla 9.3 Tipos de datos de HL7 que representan códigos

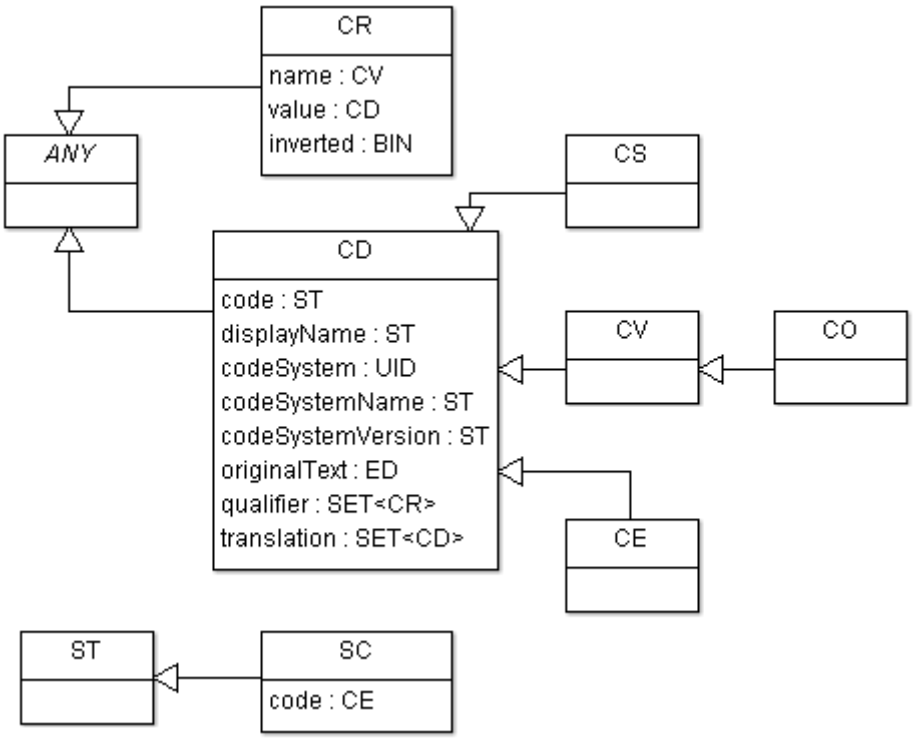


Figura 9.3 Diagrama de los tipos código de HL7

9.4 Nombres, identificadores y localizaciones

En la tabla 9.4, se pueden observar los tipos de datos relativos a nombres, identificadores y localizaciones.

Tipo	Descripción
II (*Instance Identifier*)	Identificadores que se utilizan para identificar de forma unívoca un objeto.
URL (*Universal Resource Locator*)	Dirección electrónica especificada según el *Internet Standard RFC 2396*
TEL (*Telecommunication Address*)	Cualquier localizador dentro del campo de las telecomunicaciones como números de teléfono, de fax y direcciones de correo electrónico.
ADXP (*Address Part)*	Parte de una dirección postal como puede ser el número, el piso o la escalera.
ENXP (*Entity Name Part)*	Cadena de caracteres que representa una parte de un nombre completo, como los apellidos de una persona.
UID (*Unique Identifier String)*	Cadena de caracteres que representa un código único.
UUID (*Universal Unique Identifier*)	Cadena de caracteres que representa un código único en un formato llamado UUID, que consta de 5 grupos de dígitos hexadecimales separados por el carácter '-'.
OID (*Object Identifier*)	Cadena de caracteres que representa un código único en un formato llamado OID, que consta solamente de números y puntos.
AD (*Postal Address*)	Direcciones postales de hogares y oficinas.
EN (*Entity Name*)	Nombre de persona, organización, lugar o cosa.
TN (*Trivial Name*)	Nombre de un lugar o de una cosa.
ON (*Organization Name*)	Nombre de organización.
PN (*Person Name*)	Nombre de persona.

Tabla 9.4 Tipos de datos relativos a nombres, identificadores y localizaciones

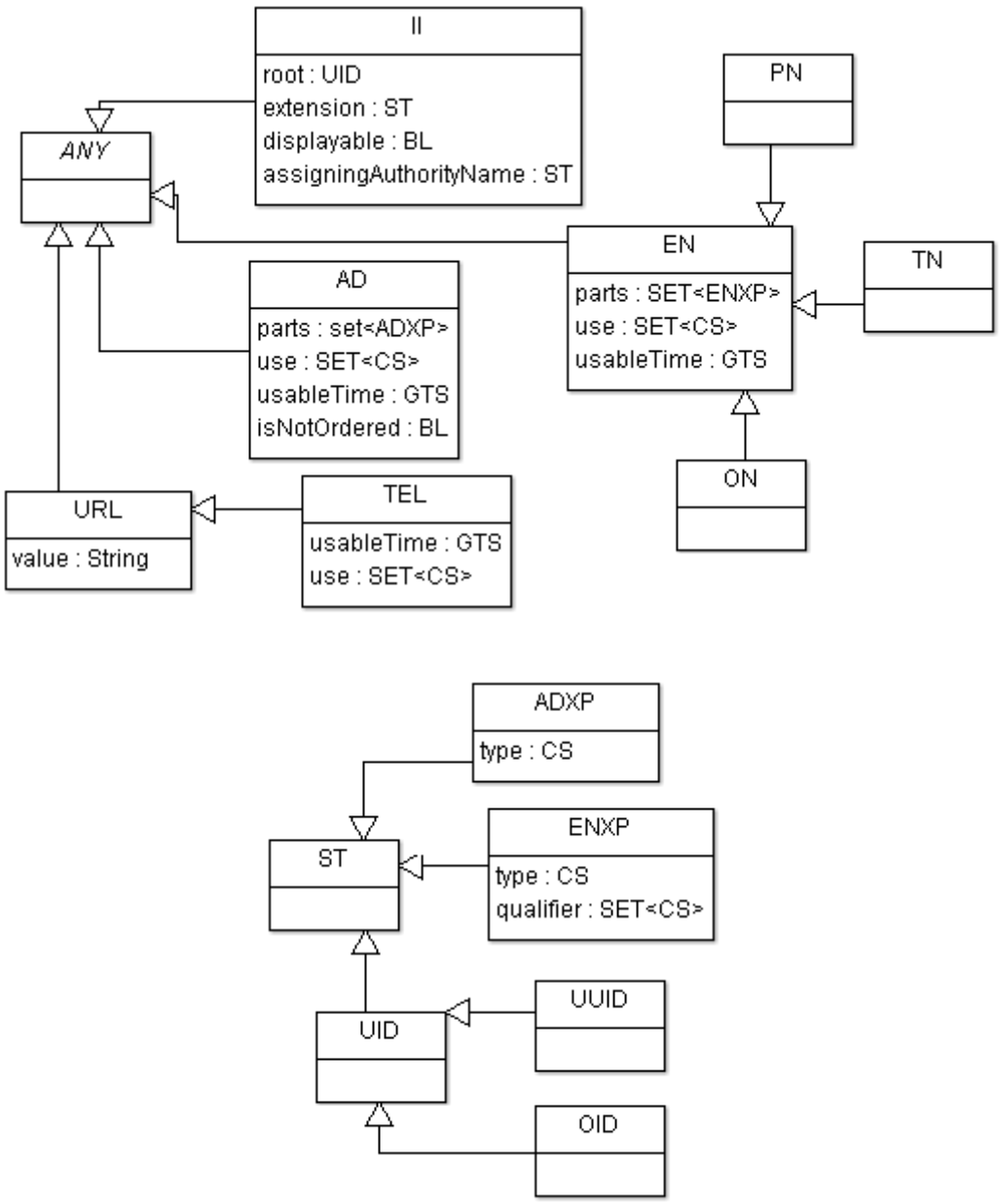


Figura 9.4 Diagrama de los nombres, identificadores y localizaciones de HL7

9.5 Cantidades

En la tabla 9.5, se muestran los tipos de datos que expresan cantidades.

Tipo	Descripción
QTY (*Quantity*)	Generalización de todos los tipos que aparecen a continuación en esta misma tabla.
INT (*Integer*)	Número entero.

RTO (*Ratio*)	Cantidad resultante de la división de un par de cantidades.
REAL (*Real*)	Número real.
MO (*Monetary Amount*)	Cantidad monetaria.
TS (*Point in Time*)	Momento concreto.
PQ (*Physical Quantity*)	Cantidad representada en un determinado sistema de unidades.
PQR (*Physical Quantity Representation*)	Representa una cantidad en un sistema de unidades definido a partir de un código.

Tabla 9.5 Tipos de datos de HL7 que expresan cantidades

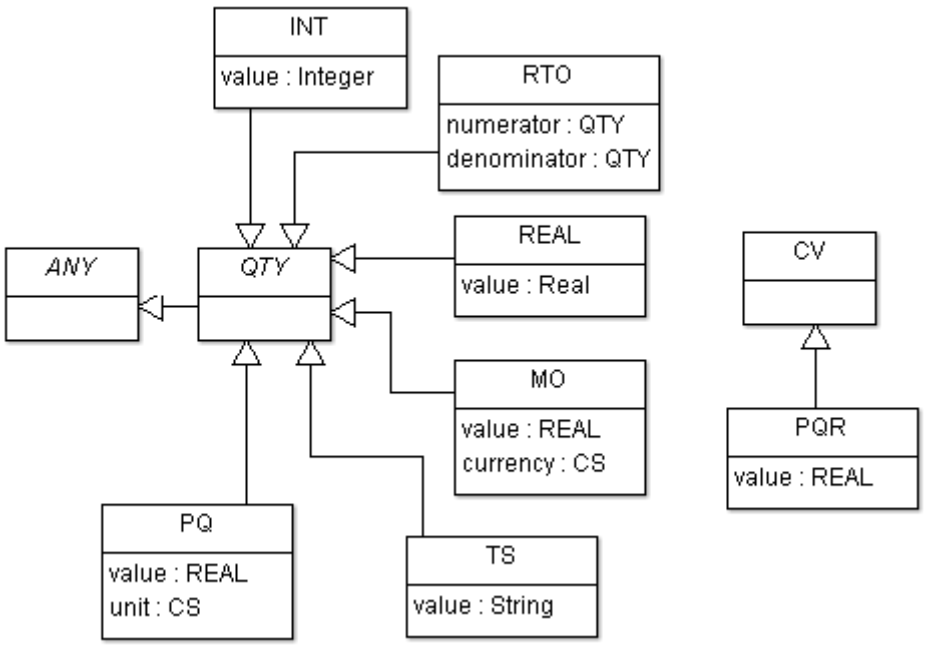


Figura 9.5 Diagrama de los tipos de cantidades en HL7

10 MECANISMOS DE REFINAMIENTO

Se han presentado los distintos modelos de información y señalado de cuáles derivan cada uno de ellos, pero hasta ahora no se han explicado los mecanismos que nos permiten, por ejemplo, obtener un R-MIM a partir de un D-MIM.

En este capítulo, se describen esos mecanismos, que básicamente consisten en aplicar una serie de restricciones que nos permiten pasar de un modelo general a uno más específico. Todas esas restricciones pueden clasificarse en uno de los siguientes grupos: omisión, clonado, restricción de cardinalidades, restricciones respecto a los tipos de datos y restricción en cuanto a los códigos.

10.1 Omisión

Constituye la forma más simple de refinamiento. Simplemente consiste en eliminar clases, atributos y otros elementos que aparecían en el modelo original y que no resultan de interés en el ámbito en el que aplica el nuevo.

Un caso donde se aprecia claramente la aplicación de este método, se da cuando en el modelo original se tiene un *choice* que incluye un número elevado de elementos y se eliminan todos aquellos que no aplican en el dominio del nuevo. Así por ejemplo, en el D-MIM con identificador *PRPA_DM000000UV*, aparece el *choice* que se muestra en la figura 10.1, mientras que en uno de los R-MIM que derivan de él, en concreto el que tiene identificador *PRPA_RM101301UV*, sólo aparece una de las dos clases que contiene el *choice*, *Person*.

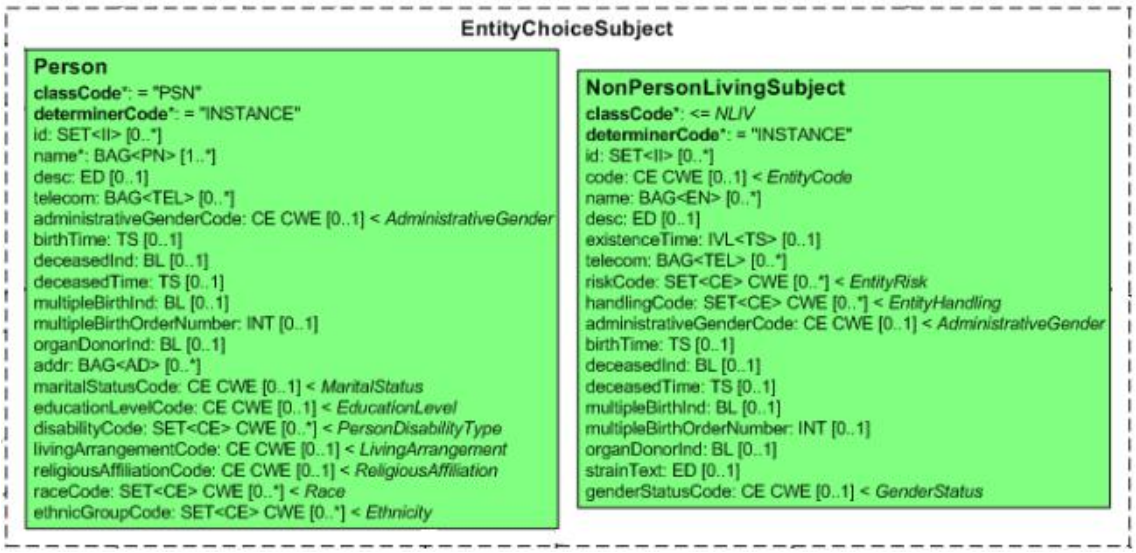


Figura 10.1 Choice EntityChoiceSubject, obtenido de [Hl7Ball]

10.2 Clonado

Consiste en hacer que una determinada clase que aparecía una única vez en el modelo original, aparezca múltiples veces en el derivado. Resulta sencillo comprobar que se utiliza esta técnica, puesto que, por ejemplo, en el RIM no existe ninguna clase repetida, pero tal y como se ha explicado anteriormente, en algunos D-MIM y R-MIM aparecen clases repetidas y son fácilmente identificables ya que no aparecen coloreadas de forma uniforme como las demás.

10.3 Restricción de cardinalidades

Al aplicar esta técnica, se consigue hacer más restrictivas las cardinalidades referentes a atributos o asociaciones. En definitiva, esto significa que el rango definido por la nueva cardinalidad mínima y máxima, debe estar contenido en el rango de la anterior. Así por ejemplo, es posible derivar un R-MIM que contenga un determinado atributo con una cardinalidad 1..2, de un D-MIM que fijase una cardinalidad de 1..* para ese mismo atributo.

10.4 Restricciones respecto a los tipos de datos

Los tipos de datos de HL7 están organizados siguiendo una estructura jerárquica. Lo que se permite hacer al derivar un modelo, es pasar de un tipo de dato más general a uno más específico, es decir, a uno que se encuentre más abajo en la jerarquía definida. Así por

ejemplo, si en un determinado D-MIM se tenía un atributo del tipo *QTY* (Cantidad), en un R-MIM que se derive de él, ese atributo podría ser del tipo *INT* (Entero), que es más específico y en la jerarquía está un nivel por debajo de *QTY*.

10.5 Restricción en cuanto a los códigos

Anteriormente, se ha explicado que los tipos de datos que corresponden a códigos pueden considerarse CWE (*Coded with exceptions*) o CNE (*Coded no exceptions*). En el caso que en el modelo original se tenga CNE, en el nuevo también deberá ser así, en caso contrario, podrá ser CWE o CNE. Esto es completamente lógico, pues significa que cuando en el modelo original se tiene un código que pertenece a un dominio concreto fijado, en el nuevo no se puede usar uno arbitrario.

11 EJEMPLOS HL7

Para cerrar el bloque dedicado al estándar HL7, se muestran algunos ejemplos donde se pueden observar conjuntamente los elementos descritos a lo largo del mismo.

11.1 D-MIM y R-MIM

En primer lugar, se explica y se muestra un subconjunto del modelo que representa el D-MIM perteneciente al dominio de planificación (*Scheduling*), extraído del *ballot* de mayo de 2009 (figura 11.1).

La clase foco del dominio, es *ActAppointment*, que representa un acto que se está planificando (visita médica, intervención quirúrgica, etc.). En primer lugar, se puede observar que esta clase está relacionada con el acto *ActAppointmentRequest*, que representa la petición que causa que el acto actual se tenga que planificar.

Se puede ver que en la planificación del acto que representa *ActAppointment*, se tiene en cuenta que hay alguien que juega el rol de paciente (*R_Patient*) y que es el sujeto (*Subject*) del acto, una persona asignada (*R_AssignedPerson*) que participará como realizadora del acto (*performer*) y una localización (*R_ServiceDeliveryLocation*), que participa como lugar donde ocurre el acto representado (*Location*). Se puede comprobar que en estos tres casos, los roles aparecen representados mediante CMETs, esto es debido a que, como se puede intuir, los roles de paciente, personas asignadas y localizaciones, aparecen en varios modelos distintos y por tanto, como se ha explicado anteriormente, para no repetir los mismos elementos en múltiples esquemas se opta por usar CMETs.

Otro de los elementos que se tienen en cuenta en la planificación del acto son los dispositivos (*Device*) que intervienen en él. Aquí se puede volver a apreciar el uso de los roles *scoper* y *player* explicados con anterioridad. El que juega el rol es el dispositivo y por tanto es el *player*, asimismo, dichos dispositivos pertenecen a una organización determinada (*E_Organization*), por esto, esta última tendrá el rol de *scoper*.

Cabe destacar que todas las asociaciones existentes respetan aquellas definidas en el RIM. Es decir, las clases de tipo *Act* se relacionan entre ellas mediante una de tipo *ActRelationship*, una de tipo *Participation* relaciona *Acts* con *Roles*, etc.

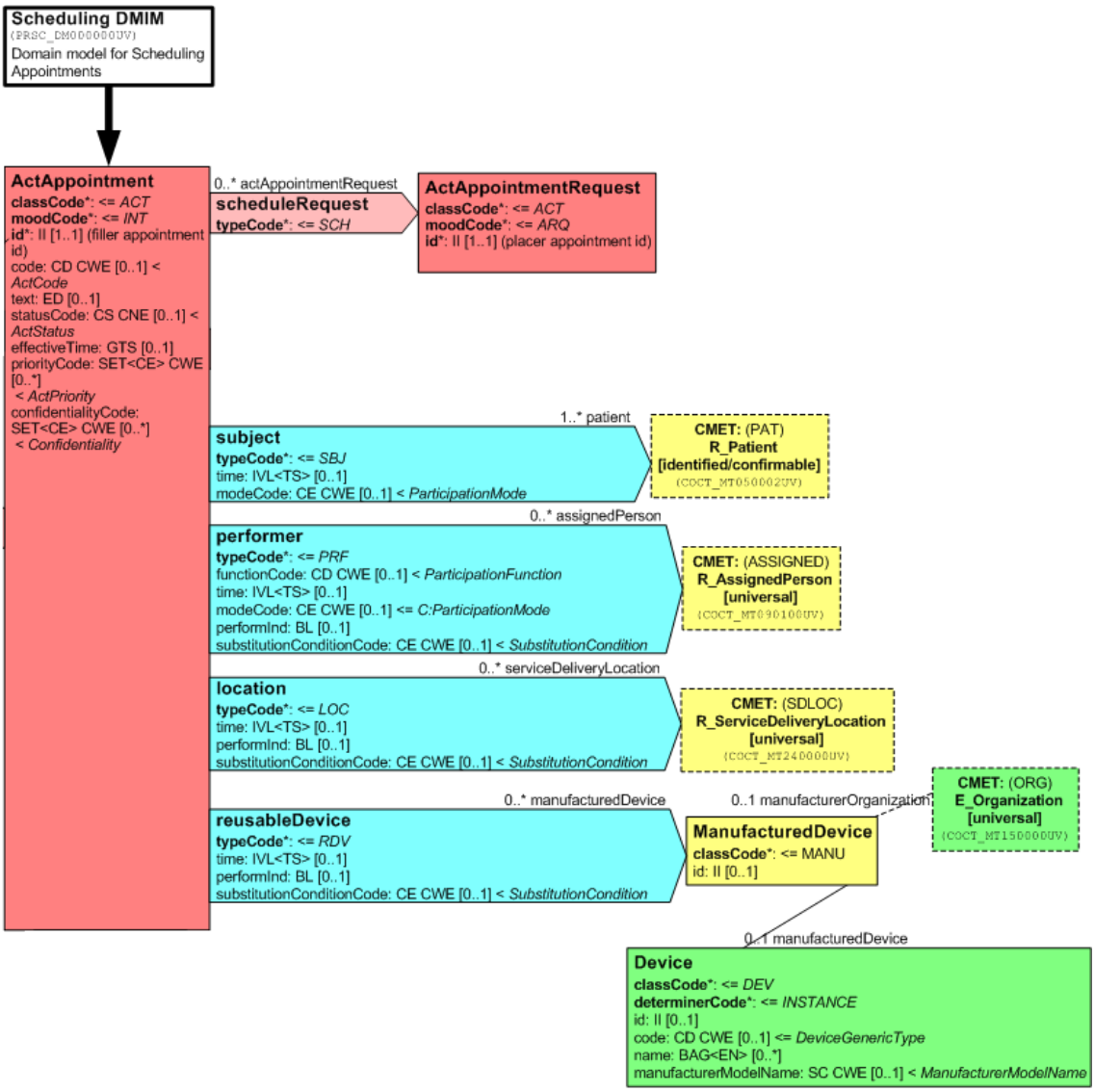


Figura 11.1 Subconjunto del D-MIM del dominio Scheduling

Se ha decidido no mostrar un ejemplo de R-MIM, puesto que contienen un subconjunto de un D-MIM y por tanto, no aportaría nada nuevo. No obstante, se ofrecen fragmentos de un HMD, de un XSD y de un mensaje XML para que el lector pueda hacerse una idea de su estructura, ya que si bien es cierto que en el proyecto que nos ocupa no resultan especialmente relevantes, también lo es que son una parte destacable dentro del estándar.

11.2 HMD

El fragmento de HMD que se ofrece en la figura 11.2, ha sido obtenido a partir de un R-MIM derivado del D-MIM ilustrado anteriormente y se ha extraído del *ballot* de mayo de 2009 en su versión HTML.

En el fragmento mostrado, se pueden observar las clases *Subject, ReusableDevice, ManufacturedDevice y Device* derivadas del D-MIM anterior, junto a sus atributos y asociaciones, que aparecen sobre un fondo gris en la tabla. En cuanto a estas últimas, cabe destacar que en el HMD, sólo aparecen representadas en un sentido, que corresponde al que marcan las flechas en los D-MIM y R-MIM. Así por ejemplo, consultando la clase *ManufacturedDevice* en el siguiente HMD, podemos saber que está relacionada con *Device*, pero consultando sólo la clase *Device*, no podemos conocer la existencia de dicha asociación.

Subject
typeCode [1..1] (M) Participation (CS) {CNE:V:**ParticipationTargetSubject**, root= "SBJ"}
time [0..1] Participation (IVL<TS>)
modeCode [0..1] Participation (CE) {CWE:D:**ParticipationMode**, default= "PHYSICAL"}
patient [1..1] (R_PatientIdentified-confirmable)

ReusableDevice
typeCode [1..1] (M) Participation (CS) {CNE:V:**ParticipationReusableDevice**, root= "RDV"}
time [0..1] Participation (IVL<TS>)
performInd [0..1] Participation (BL)
substitutionConditionCode [0..1] Participation (CE) {CWE:D:**SubstitutionCondition**}
manufacturedDevice [1..1] (ManufacturedDevice)

ManufacturedDevice
classCode [1..1] (M) Role (CS) {CNE:V:**RoleClassManufacturedProduct**, root= "MANU"}
id [0..1] Role (II)
manufacturedDevice [0..1] (Device)
manufacturerOrganization [0..1] (E_OrganizationUniversal)

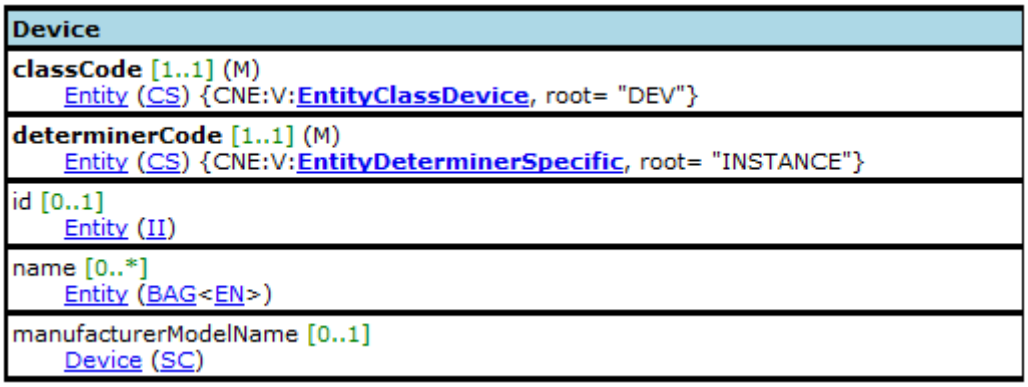


Figura 11.2 Fragmento de HMD del D-MIM del dominio de Scheduling, obtenido de [Hl7Ball]

11.3 XSD

En el ejemplo 11.1, se muestra un fragmento del fichero XSD derivado del HMD anterior. Concretamente, se muestran los trozos de código correspondientes a las clases *ReusableDevice*, *ManufacturedDevice* y *Device* del ejemplo anterior. A partir de este fichero XSD, se definen mensajes XML que se pueden intercambiar entre distintos sistemas que implementen el estándar HL7.

```
<xs:complexType name="PRSC_MT010101UV01.ReusableDevice">
  <xs:sequence>
    <xs:group ref="InfrastructureRootElements"/>
    <xs:element name="time" type="IVL_TS" minOccurs="0"
      maxOccurs="1"/>
    <xs:element name="performInd" type="BL" minOccurs="0"
      maxOccurs="1"/>
    <xs:element name="substitutionConditionCode" type="CE"
      minOccurs="0" maxOccurs="1"/>
    <xs:element name="manufacturedDevice"
      type="PRSC_MT010101UV01.ManufacturedDevice"
      nillable="true"
      minOccurs="1"
      maxOccurs="1"/>
  </xs:sequence>
  <xs:attributeGroup ref="InfrastructureRootAttributes"/>
  <xs:attribute name="nullFlavor" type="NullFlavor" use="optional"/>
  <xs:attribute name="typeCode" type="ParticipationType"
    use="required" fixed="RDV"/>
</xs:complexType>

<xs:complexType name="PRSC_MT010101UV01.ManufacturedDevice">
  <xs:sequence>
    <xs:group ref="InfrastructureRootElements"/>
    <xs:element name="id" type="II" minOccurs="0" maxOccurs="1"/>
    <xs:element name="manufacturedDevice"
      type="PRSC_MT010101UV01.Device" nillable="true"
      minOccurs="0"
      maxOccurs="1"/>
    <xs:element name="manufacturerOrganization"
```

```
      type="COCT_MT150000UV02.Organization"
      nillable="true"
      minOccurs="0"
      maxOccurs="1"/>
  </xs:sequence>
  <xs:attributeGroup ref="InfrastructureRootAttributes"/>
  <xs:attribute name="nullFlavor" type="NullFlavor" use="optional"/>
  <xs:attribute name="classCode" type="RoleClassManufacturedProduct"
    use="required"/>
</xs:complexType>

<xs:complexType name="PRSC_MT010101UV01.Device">
  <xs:sequence>
    <xs:group ref="InfrastructureRootElements"/>
    <xs:element name="id" type="II" minOccurs="0" maxOccurs="1"/>
    <xs:element name="name" type="EN" minOccurs="0"
      maxOccurs="unbounded"/>
    <xs:element name="manufacturerModelName" type="SC" minOccurs="0"
      maxOccurs="1"/>
  </xs:sequence>
  <xs:attributeGroup ref="InfrastructureRootAttributes"/>
  <xs:attribute name="nullFlavor" type="NullFlavor" use="optional"/>
  <xs:attribute name="classCode" type="EntityClassDevice"
    use="required"/>
  <xs:attribute name="determinerCode" type="EntityDeterminer"
    use="required" fixed="INSTANCE"/>
</xs:complexType>
```

Ejemplo 11.1 Fragmento de fichero XSD del D-MIM del dominio de Scheduling

11.4 Mensaje XML

En el ejemplo 11.2, se muestra el fragmento de un posible mensaje XML que se intercambiaría entre dos sistemas y que contendría la información relativa a las clases *ReusableDevice*, *ManufacturedDevice* y *Device* que han aparecido en los ejemplos anteriores.

Para componer el mensaje correctamente, hay que tener en cuenta que todos aquellos atributos obligatorios deben aparecer, y que aquellos que son de tipo código, como el typeCode y el classCode, que determinan la clase que los contiene, no deben definirse puesto que su valor queda implícito.

```
<reusableDevice>
  <time value="20101117124128"/>
  <ManufacturedDevice>
    <id root="1" extension="12345"/>
    <Device>
     <id root="1" extension="1234"/>
      <name>Device 1</name>
    </Device>
  </ManufacturedDevice>
</reusableDevice>
```

Ejemplo 11.2 Fragmento de mensaje XML relativo al dominio de Scheduling

HERRAMIENTA DE TRANSFORMACIÓN DE MODELOS

12 ESPECIFICACIÓN

En el bloque que se inicia con este capítulo, se explica todo lo relacionado con la herramienta de transformación de modelos. Esto incluye su proceso de desarrollo, los problemas que presenta el estándar HL7 y que lo han dificultado, ejemplos de conversiones, guías tanto a nivel de usuario como a nivel de administrador, un pequeño estudio sobre el rendimiento de la aplicación y finalmente cómo se ha usado la herramienta UmlCanvas para hacer que los esquemas resultantes más accesibles.

En este capítulo concreto, se describe el proceso de desarrollo del software que transforma modelos de HL7 en modelos UML. Dicho proceso, consta de diversas fases claramente diferenciadas. Con el fin de poder exponerlo de forma clara, se ha optado por hacer una breve introducción, donde se describen de forma resumida cada una de estas fases y más adelante, ofrecer un capítulo dedicado a cada una de ellas, donde se presentan detalladamente.

En la figura 12.1, se muestra un esquema que permite distinguir con claridad todos los elementos que intervienen en el desarrollo de la herramienta. A partir de dichos elementos, resulta sencillo distinguir cuáles son las fases que componen el proyecto.

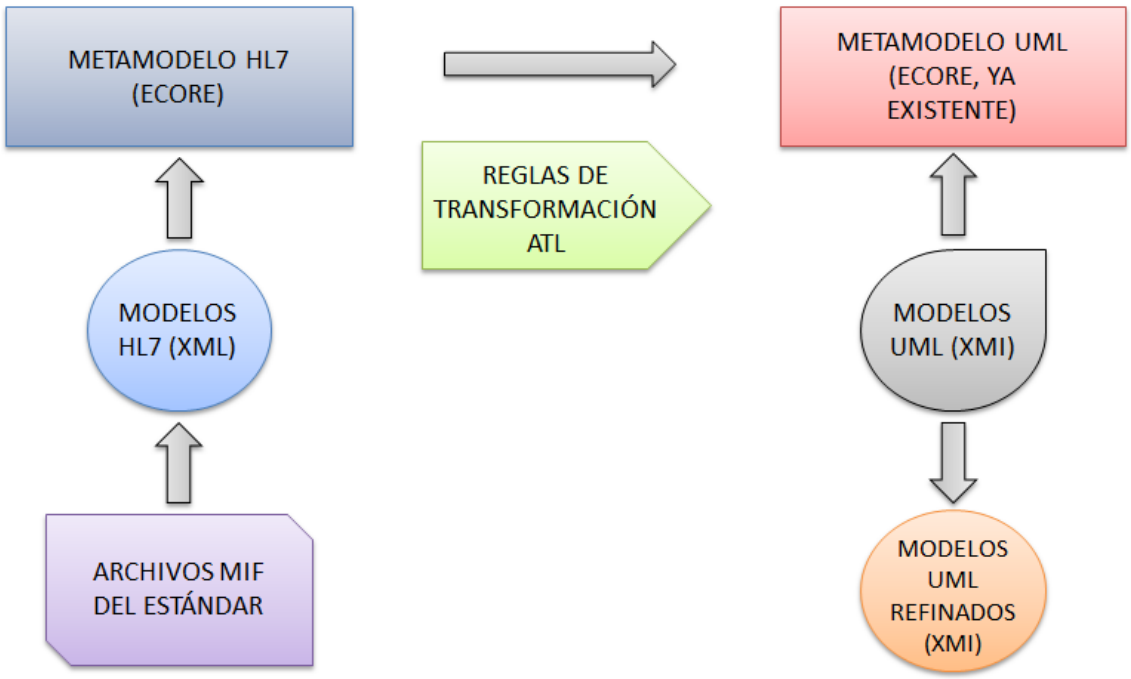


Figura 12.1 Esquema general de la herramienta de conversión de modelos

A continuación, se procede a detallar los elementos que aparecen en el esquema anterior.

Metamodelo HL7. Los modelos disponibles en el estándar HL7, se rigen por toda una serie de normas y restricciones definidas implícitamente. El primer paso que debemos dar, consiste en diseñar un metamodelo que refleje todas esas características y del cual, todos los modelos disponibles en el estándar se puedan considerar instancias. El formato que se ha escogido para representarlo es el *Ecore*, el lenguaje propio de la herramienta *Eclipse* para definir metamodelos. Como se verá en el capítulo dedicado al metamodelo de HL7, el diseñarlo nos ayudará tanto a la hora de definir las reglas que nos permiten pasar de HL7 a UML, como al tratar los ficheros del estándar que contienen la información de los modelos HL7.

Archivos MIF del estándar. La información de los modelos del estándar HL7, se encuentra disponible en distintos formatos. En el proyecto que nos ocupa, se ha optado por utilizar los archivos MIF (*Model Interchange Format*). En su capítulo correspondiente, se explicarán las distintas opciones que se han valorado en el desarrollo del proyecto y se justificarán las decisiones tomadas.

Modelos HL7. Se necesita tratar los archivos MIF, para a partir de ellos, poder generar instancias escritas en XML del metamodelo HL7 anteriormente descrito. Para esta tarea, se ha optado por desarrollar un *parser* codificado en Java. El hecho de tener el metamodelo descrito en *Ecore*, nos facilita esta tarea de forma considerable, puesto que *Eclipse* a partir de él, es capaz de generar una API que nos permite tratar de una forma simple con los elementos que aparecen en dicho metamodelo.

Metamodelo UML. Este metamodelo, resulta necesario para definir las reglas de conversión correctamente, ya que el objetivo consiste en que los modelos resultantes sean instancias suyas, o lo que es lo mismo, que sean válidos según la especificación de UML. Este metamodelo no lo hemos tenido que diseñar nosotros mismos. Como se ha explicado anteriormente, la *Object Management Group* (OMG), organización que promueve diversos estándares entre los que se encuentran el UML, se encarga del diseño de dicho metamodelo, y diversos grupos de desarrollo de *Eclipse*, de generar su versión en formato *Ecore*.

Reglas de transformación ATL. ATL es una herramienta de transformación de modelos construida sobre *Eclipse*, que permite, dados dos metamodelos MM1 y MM2, una instancia de MM1 escrita en XML y una serie de reglas de conversión de MM1 hacia MM2, generar una instancia de MM2 también escrita en XML, la cual contiene la información transformada de la instancia de MM1 una vez se aplican las reglas para pasar de MM1 a MM2. Este conjunto de reglas constituye la parte central del proyecto, pues es donde se definen las transformaciones necesarias para obtener los modelos UML.

Modelos UML. Al final del proceso, obtenemos un archivo XMI que es instancia del metamodelo UML, o dicho de otra manera, un modelo válido según las reglas definidas por UML, y que contiene la misma información que el archivo MIF de origen.

Modelos UML refinados. Existen ciertas características que nos gustaría que los UML resultantes incorporasen, pero que no podemos obtener simplemente aplicando las reglas ATL. Estos aspectos tienen que ver, principalmente, con la forma de referenciar elementos que se encuentran definidos en otros ficheros UML. Por este motivo, resulta necesario desarrollar una pequeña aplicación que corrija estos pequeños detalles, generando nuevos ficheros UML a partir de los obtenidos al aplicar las reglas de transformación.

A partir de los elementos descritos, resulta lógico dividir el trabajo realizado en el desarrollo de la herramienta en cuatro fases claramente diferenciables:

- En la primera de ellas, tiene cabida todo aquello relacionado con el diseño del metamodelo de HL7.
- En la segunda, se encuentran los aspectos que tienen que ver con el trato de los archivos MIF y su conversión a un archivo XML que sea instancia del metamodelo HL7 diseñado.
- En la tercera, se trata la definición de las reglas de conversión de la herramienta ATL.
- En la cuarta y última, es en la que se refinan los UML que se obtienen al aplicar las reglas ATL.

En la figura 12.2, se muestra un esquema en el que se puede apreciar cómo se relacionan estas cuatro fases entre ellas. Además, se pueden distinguir las entradas que reciben y las salidas que generan.

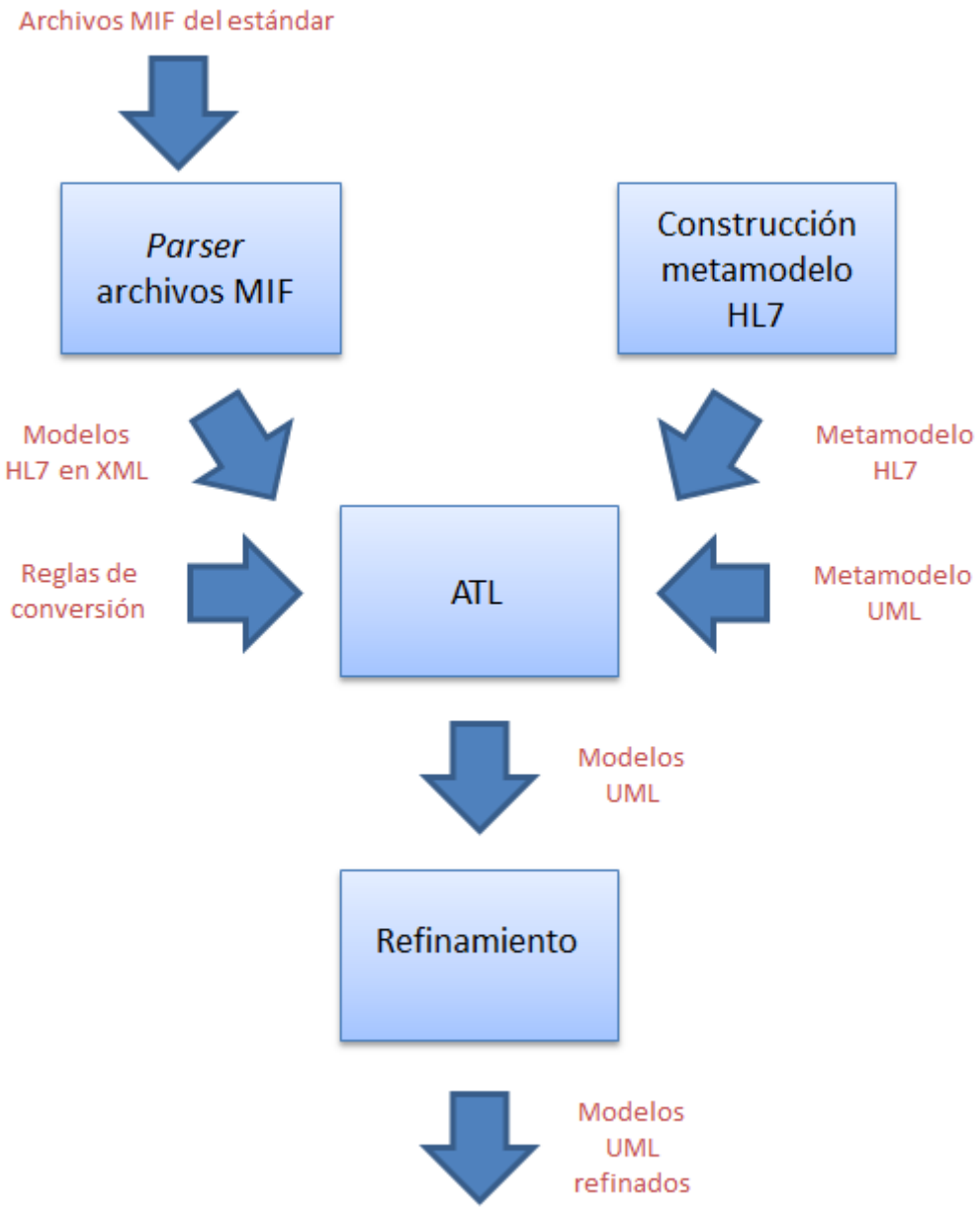


Figura 12.2 Fases del proyecto

A continuación, tal y como se ha afirmado anteriormente, se procede a explicar cada una de las cuatro etapas de desarrollo en un capítulo propio.

13 METAMODELO DE HL7

En este capítulo, se expone todo aquello relacionado con el diseño del metamodelo de HL7. Dicho metamodelo no se puede encontrar de forma explícita en el estándar HL7. Es decir, no encontramos ningún documento o esquema único donde se describan todos los elementos que lo componen. No obstante, analizando los distintos esquemas conceptuales que ofrece y la gran cantidad de documentos donde se explican detalles de sus elementos, resulta viable deducirlo. Esto es, precisamente, lo que se ha hecho en este proyecto y en este mismo capítulo, se justificarán todas las decisiones de diseño tomadas.

Por otro lado, la necesidad de diseñar este metamodelo, resulta fácilmente justificable por tres razones:

- La primera es que resulta imprescindible para poder utilizar la herramienta de transformación de modelos ATL.
- La segunda es que a partir de él, *Eclipse* es capaz de generar automáticamente una API que nos ayuda a tratar todos los archivos MIF del estándar, que constituyen la fuente a partir de la cual se extrae la información de los modelos.
- Por último, resulta muy útil de cara a poder asegurarnos nosotros mismos de que entendemos todos los elementos que componen el estándar y las relaciones que se dan entre ellos, ya que a partir de la instanciación de modelos, podremos comprobar si se ha definido correctamente o no.

En la sección que se presenta a continuación, se explica cómo se ha diseñado el metamodelo, analizando cada uno de los elementos que se incluyen en el estándar de forma separada. También se proporcionan explicaciones sobre las herramientas usadas para definirlo y algunos ejemplos de instanciaciones de esquemas, que nos permitirán garantizar que se ha definido correctamente.

13.1 Diseño del metamodelo

Con el objetivo de hacer que el proceso de diseño del metamodelo que se ha seguido resulte más sencillo de entender, se ha optado por exponer elemento por elemento sus propiedades y relaciones. Además, también se presentan en lenguaje natural las restricciones de integridad que afectan a cada uno de ellos. Más adelante, en concreto en la sección 13.2.2, se describen

en OCL. El motivo es que la descripción formal resulta mucho más sencilla de explicar una vez se han detallado todos los elementos.

Cabe destacar, que algunos de esos elementos son muy parecidos o iguales a los definidos en el metamodelo de UML descrito en el capítulo 5 de este documento. Por este motivo, resulta conveniente estar familiarizado con este último de cara a poder comprender las explicaciones proporcionadas. Del mismo modo, también se hace necesario haber leído el capítulo 8, en el que se encuentran las explicaciones detalladas de todos los elementos que componen el estándar.

13.1.1 Elementos del metamodelo

Los elementos descritos en este apartado, son los mismos que fueron presentados en el capítulo 8 dentro del bloque dedicado al estándar HL7. Concretamente son: clases, atributos, asociaciones, restricciones, notas, *choices*, CMETs, *Entry Points*, los tipos de datos y *Generalization*.

13.1.1.1 Elementos

Tal y como sucede en el metamodelo de UML, en el de HL7 se ha optado por utilizar las metaclases *Element*, *NamedElement* y *TypedElement*, para organizar el resto de metaclases de forma jerárquica.

Element representa cualquier elemento, *NamedElement* todos aquellos que poseen un nombre, y por último, *TypedElement* todos aquellos que además de un nombre, tienen un tipo. Todas ellas son abstractas puesto que no tiene sentido que existan instancias de estas metaclases, las instancias deben de ser de sus subclases.

La justificación de este hecho es simple, hay muchos elementos que poseen un nombre y todos ellos presentan algunas asociaciones comunes. Lo que se consigue creando la metaclase *NamedElement* es, por una parte no repetir el atributo *name* en todas ellas, y por otra, no repetir asociaciones de forma innecesaria. Con los elementos con tipo, ocurre exactamente lo mismo.

El hecho que un diagrama contenga atributos o asociaciones repetidas, no afecta a su semántica, pero sí que tiene un gran impacto en su claridad. Es un problema que en el

metamodelo de UML se resuelve de forma elegante y que en nuestro caso, constituye una solución perfectamente válida, por lo que la mejor opción consiste en adoptarla.

En la figura 13.1, se puede apreciar la jerarquía definida entre estas tres clases.

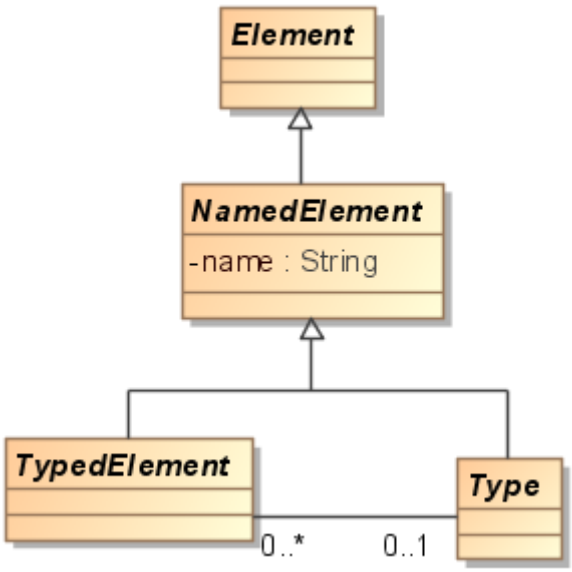


Figura 13.1 Elementos en el metamodelo de HL7

13.1.1.2 Clases

Las clases presentes en los esquemas conceptuales de HL7, al igual que ocurre con las clases UML, se usan para representar conceptos del mundo real. La única distinción está, en que las clases de HL7 no presentan operaciones, tan sólo atributos. Salvando esta pequeña diferencia, se puede afirmar que son elementos equivalentes.

Recordemos que en HL7 existen siete tipos de clases distintos: *Act*, *ActRelationship*, *Participation*, *Role*, *RoleLink*, *Entity* y *Infraestructure*. A la hora de representar este hecho en el metamodelo, se nos presentan principalmente dos posibilidades, que son:

- Representar cada uno de los tipos en una metaclase separada y que éstas hereden de la metaclase abstracta *Class*. (Figura 13.2).
- Dotar a la metaclase *Class* de un atributo *type* que sea del tipo enumerativo *HL7Type* y que éste contengan los siete citados anteriormente. (Figura 13.3).

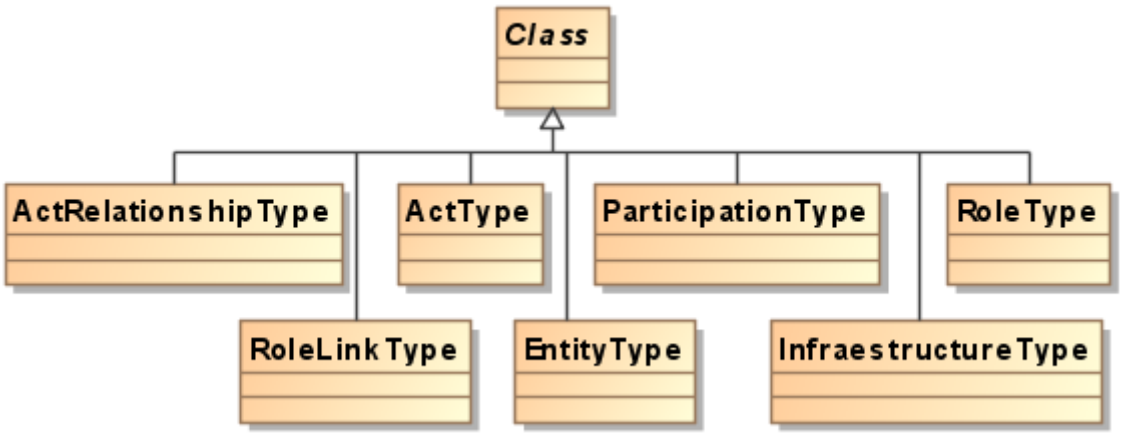


Figura 13.2 Jerarquía de Clase en el metamodelo de HL7

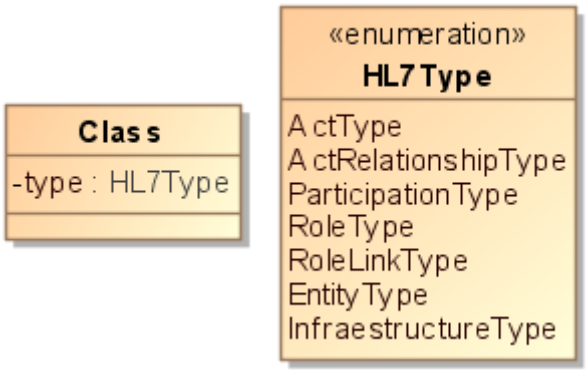


Figura 13.3 Opción de representar Clase en el metamodelo HL7 desestimada

Nos hemos decantado por la primera opción por dos razones. La primera es que los distintos tipos de clase hacen referencia a elementos claramente diferenciables entre sí. Un concepto representado por una clase de tipo *Role* como paciente o cirujano, es una realidad completamente distinta de los elementos representados por clases de tipo *Entity* como pueden ser, por ejemplo, personas u organizaciones.

La segunda razón, es que pese a que cada una de las clases no presenta atributos propios, sí que es cierto que las asociaciones están regidas por restricciones diferentes. Por ejemplo, recordemos que una clase de tipo *ActRelationship*, sólo puede estar relacionada con otras de tipo *Act*, mientras que, por ejemplo, una de tipo *Participation*, puede estar relacionada tanto con clases de tipo *Act* como de tipo *Role*. Cuando se da este caso, a pesar de que las dos soluciones anteriormente expuestas sean equivalentes desde un punto de vista semántico, la primera se suele considerar más elegante.

13.1.1.3 Atributos

Los atributos que contienen las clases de HL7, presentan toda una serie de propiedades (*domainName*, *codeSystemName*, *mnemonic*), que se refieren al vocabulario relativo al atributo que los contiene. Por otro lado, también encontramos *defaultValue*, que es de tipo derivado y nos indica su valor por defecto, y por último, *codingStrength* que en caso que el atributo tenga un tipos de dato relativo a códigos, indica si es CWE (*Coded with exceptions*) o CNE (*Coded no exceptions*).

A parte de esos atributos propios, presentan las mismas características que los definidos en UML. Por este motivo, se ha optado por adaptar en nuestra solución, el fragmento del metamodelo UML donde se definen los atributos y sus elementos.

Los atributos de una clase, son instancias de la metaclase *Property* y poseen una multiplicidad representada por la metaclase *MultiplicityElement*, pueden tener un valor por defecto, que sería una instancia de la metaclase *ValueSpecification* y, finalmente, poseen un tipo, que heredan al ser subclase de *TypedElement*.

En la figura 13.4, se muestra el fragmento del metamodelo relativo a los atributos de las clases.

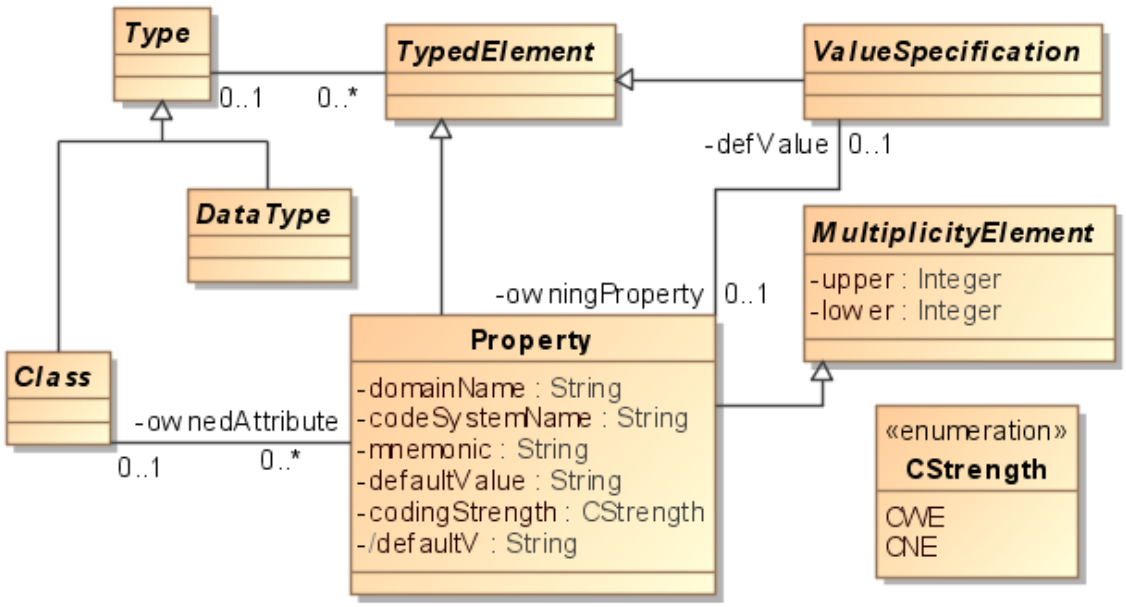


Figura 13.4 Property en el metamodelo de HL7

13.1.1.4 Asociaciones

Al igual que ocurre con los atributos, las asociaciones de HL7, presentan las mismas características que las de UML. Por ello, el fragmento de nuestro metamodelo que representa el conocimiento relativo a las asociaciones es muy similar al de UML.

No obstante, hay una diferencia que cabe destacar. En UML existen asociaciones n-arias, donde n puede tomar valores mayores o iguales que 2, mientras que en HL7 no existen asociaciones que relacionen más de dos elementos. Por este motivo, la cardinalidad del rol *memberEnd* de la asociación existente entre las metaclases *Property* y *Association*, debe ser 2 y no 2..* como ocurre en el metamodelo de UML.

Es importante recordar el doble papel que juega la clase *Property*, que representa tanto los extremos de una asociación, como los atributos de una clase.

En la figura 13.5, se encuentra el fragmento del metamodelo referido a las asociaciones entre clases HL7.

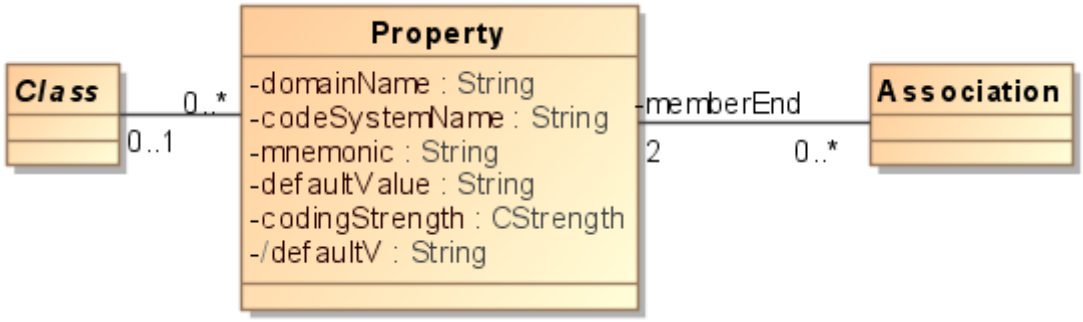


Figura 13.5 Asociación en el metamodelo de HL7

13.1.1.5 Restricciones

Las restricciones son elementos que resulta sencillo incluir en el metamodelo, puesto que tan sólo se deben tener en cuenta tres aspectos: constan de un *string* donde se define la restricción en sí, afectan a elementos y se definen en el contexto de un elemento con nombre.

En la figura 13.6, se representa la clase *Constraint*, de la cual todas las restricciones definidas son instancias, junto a las características expuestas en el párrafo anterior.

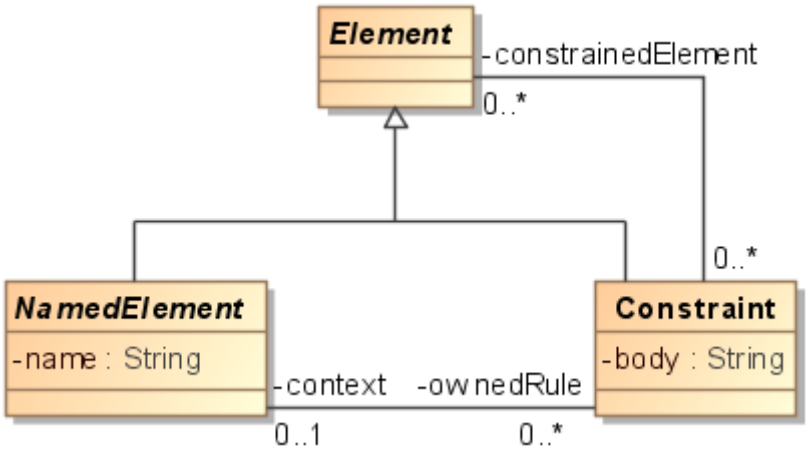


Figura 13.6 Restricción en el metamodelo de HL7

13.1.1.6 Notas

Las notas de HL7 son muy parecidas a las de UML, pero merece la pena destacar un par de aspectos que las diferencian de estas últimas. El primero es que mientras que en HL7 son conocidas como *notes*, en UML se denominan *comments*. El segundo es que se debe tener en cuenta que en HL7 existen dos tipos de notas diferenciados. Tal y como se explicó en el bloque dedicado al estándar HL7, se pueden encontrar notas descriptivas y de uso. Este último hecho, se plasma en el metamodelo haciendo que el atributo *type* de *Note* sea de tipo enumerativo.

Está claro que las notas afectan a elementos, pero para conocer las multiplicidades de los dos extremos de la asociación existente entre las metaclases *Note* y *Element*, se deben buscar ejemplos, pues a priori podríamos dudar, por ejemplo, de si en HL7 existen notas que afecten a más de un elemento al mismo tiempo. Para definir las dos multiplicidades de forma correcta, se debe conocer la respuesta a las tres preguntas siguientes, y para ello, se ha optado por buscar entre los modelos del *ballot* de septiembre de 2009.

- ¿Pueden las notas hacer referencia a más de un elemento? Sí, en el esquema *Clinical Statement Pattern*, con identificador *COCS_DM000000UV,* hay un par de notas que se refieren a las clases *sourceOf* y *targetOf* al mismo tiempo. Esta situación se muestra en la figura 13.7.

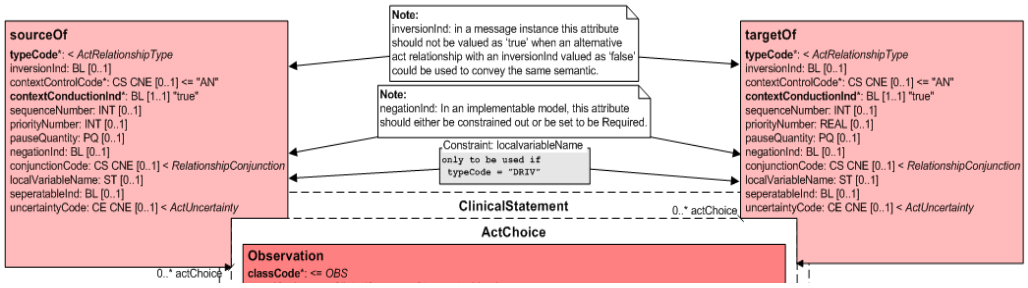


Figura 13.7 Notas HL7 que afectan a más de un elemento, obtenidas de [Hl7Ball]

- ¿Puede una nota relacionarse con cero elementos, afectando al esquema general en lugar de a elementos concretos? Sí, en el esquema *A_BillablePharmarcyDispense*, con identificador COCT_RM300000UV, se da ese caso.

- ¿Puede un mismo elemento verse referenciado por más de una nota? Sí, en el esquema *Care Provision*, con identificador *REPC_DM000000UV* (figura 13.8), hay dos notas que se refieren a la clase *CareProvision*.

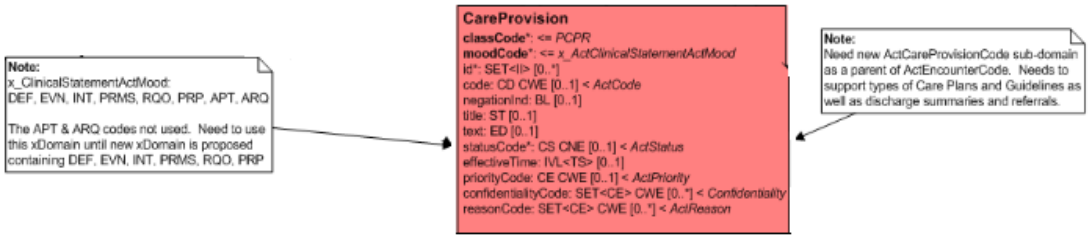


Figura 13.8 Elemento HL7 referenciado por dos notas, obtenido de [Hl7Ball]

En la figura 13.9, se puede apreciar la metaclase *Note* y su asociación con la metaclase *Element*.

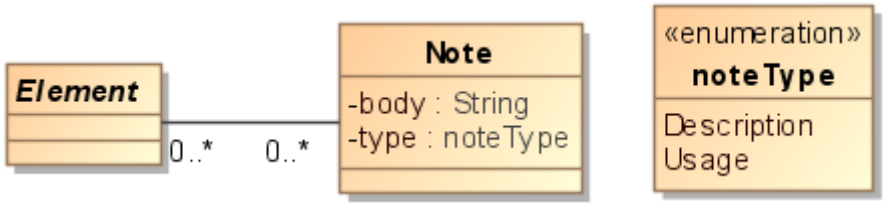


Figura 13.9 Nota en el metamodelo de HL7

13.1.1.7 Choices

Los *choices* no presentan ningún atributo. El único conocimiento que debe reflejar el metamodelo sobre ellos, consiste en saber qué elementos pueden contener.

Un *choice* puede contener otros *choices*, clases y CMETs. Por este motivo, se ha optado en el metamodelo por crear una metaclase abstracta llamada *ChoosableElement*, de la cual heredan las metaclases *Class*, *Choice* y *CMET*. Creemos que de esta manera, se representa de forma clara el carácter agrupador de los *choices*.

En cuanto a los elementos que contienen, cabe señalar que todos deben de ser del mismo tipo y que no pueden incluirse a sí mismos.

A priori, se podría tener la duda de si es posible que un mismo elemento pueda pertenecer a varios *choices* distintos. Sí que es posible y existen numerosos ejemplos, uno de ellos lo encontramos en los esquemas del *ballot* de 2010 *A_Charge universal* (*COCT_RM400000UV07*), disponible en [CharUn] y *A_Coverage basic* (*COCT_RM510005UV06*), que se halla en [CoveUn]. En el primero, encontramos que el CMET *A_Billable* está contenido en el *choice* con nombre *BillableActChoice*, y en el segundo, que pertenece al *choice BenefitChoice*.

En la figura 13.10, se encuentra el fragmento del metamodelo relativo a los *choices*.

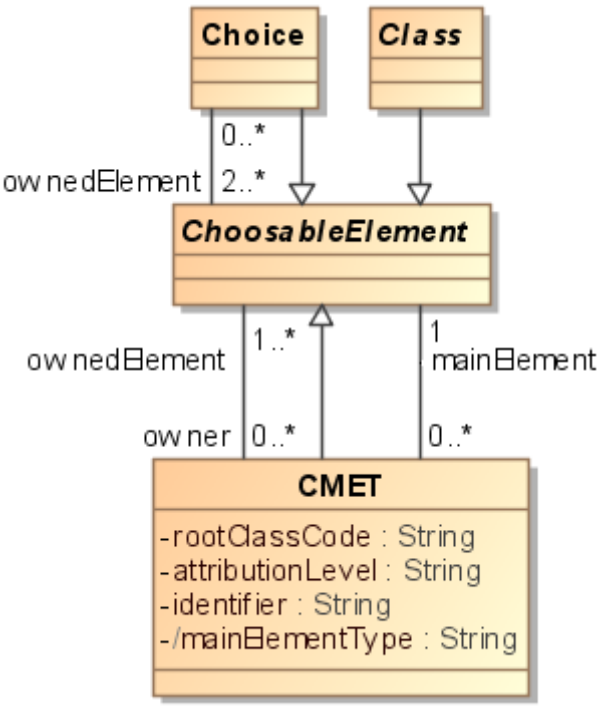


Figura 13.10 Choice en el metamodelo de HL7

13.1.1.8 CMET

Los CMETs presentan tres atributos que ya fueron explicados en un capítulo anterior: *rootClassCode*, *attributionLevel*, *identifier*, así como el atributo derivado *mainElementType*, que indica el tipo de su clase principal. Por otro lado, entre las metaclases *CMET* y *ChoosableElement*, es necesario que haya dos asociaciones, una que nos indique que elementos pertenecen a ese CMET concreto, y otra, que indique cuál de ellos es el principal.

En cuanto a las multiplicidades de la primera de esas asociaciones, puede surgir una duda similar a las de los *choices*. Debemos plantearnos si es posible que una instancia de *ChoosableElement*, pueda aparecer en más de un CMET. Y en efecto, tal y como puede apreciarse en los esquemas del *ballot* de septiembre de 2010 *A_AccountPayee basic* (*COCT_RM110202UV04*), disponible en [AccBas], y *A_AccountPayee universal* (*COCT_RM110200UV04*) , que se puede encontrar en [AccUn], es posible, ya que varias clases como *Account* y PayeeRole aparecen en ambos.

Finalmente, los CMETs se ven afectados por dos restricciones de integridad: la primera es que no puede contenerse a sí mismos, y la segunda es que su elemento principal debe encontrarse entre los elementos que contiene.

En la figura 13.11, se encuentra la parte del metamodelo que contiene el conocimiento relativo a los CMETs.

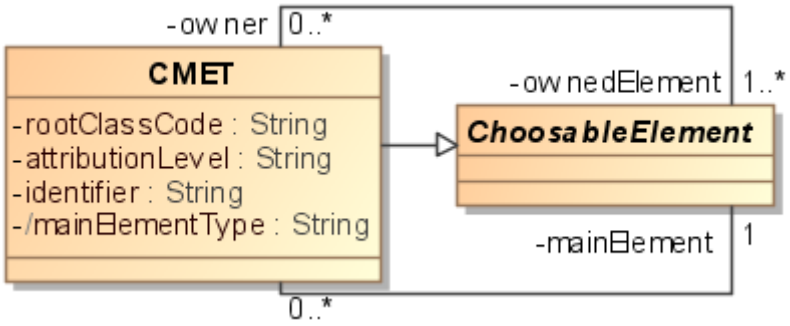


Figura 13.11 CMET en el metamodelo de HL7

13.1.1.9 Entry Point

Los *Entry Points* presentan dos atributos: un identificador y un nombre. Por otro lado, recordemos que nos indica cuál es la clase principal del esquema donde se encuentran, por tanto, se necesita una asociación entre las metaclases *EntryPoint* y *ChoosableElement*.

En el fragmento de metamodelo de la figura 13.12, encontramos representadas esas características.

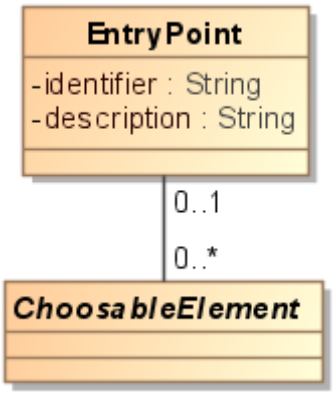


Figura 13.12 Entry Point en el metamodelo de HL7

13.1.1.10 Tipos de datos

Como ya se avanzó en el bloque dedicado a HL7, el tema de los tipos de datos es uno de los más complicados que se deben tratar en el proyecto.

Los tipos de datos HL7 son numerosos y tienen un poder expresivo mucho mayor que el de los de UML, que recordemos que sólo dispone de cuatro tipos básicos distintos: *integer*, *real*, *boolean* y *string*. En HL7, se pueden encontrar tipos de datos que, por ejemplo, indiquen cada cuanto debe tomarse un paciente un determinado medicamento, así que deben estar definidos de manera que sea posible expresar algo como “Cada día después del desayuno y de la cena”.

Desde el estándar HL7 se ha definido una especificación en UML para los tipos de datos, el problema es que los ficheros que se utilizan a nivel de implementación no reflejan exactamente lo definido a nivel de especificación. Esto, unido a la complejidad de los tipos, hace que no sea una tarea trivial el hecho de definir cuáles deben incluirse en el metamodelo de HL7 que se está diseñando.

Por ello, se ha decidido explicar con detalle toda la problemática y mostrar el fragmento de metamodelo correspondiente a los tipos de datos, en la sección 18.1 del capítulo dedicado a explicar todos los inconvenientes con los que nos hemos topado al trabajar con el estándar HL7.

13.1.1.11 Generalization

Las generalizaciones sólo se utilizan de forma implícita en el RIM, no aparecen definidas en ninguno de los otros tipos de esquemas conceptuales que se derivan de él.

Como el RIM ya está definido en UML, lo único que se debe hacer para poder representar esas generalizaciones en el metamodelo de HL7, es adoptar la misma estructura que se encuentra en el metamodelo de UML. Ésta se muestra en la figura 13.13.

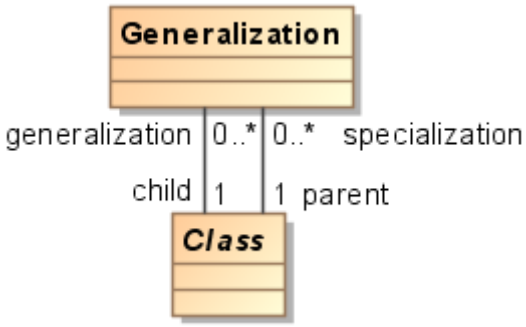


Figura 13.13 Generalización en el metamodelo de HL7

13.1.2 Metamodelo al completo

Tras haber mostrado cómo se refleja cada uno de los elementos de HL7 en el metamodelo que se ha diseñado y haberlo justificado, se procede a mostrarlo de forma íntegra, exceptuando los tipos de datos, en la figura 13.14.

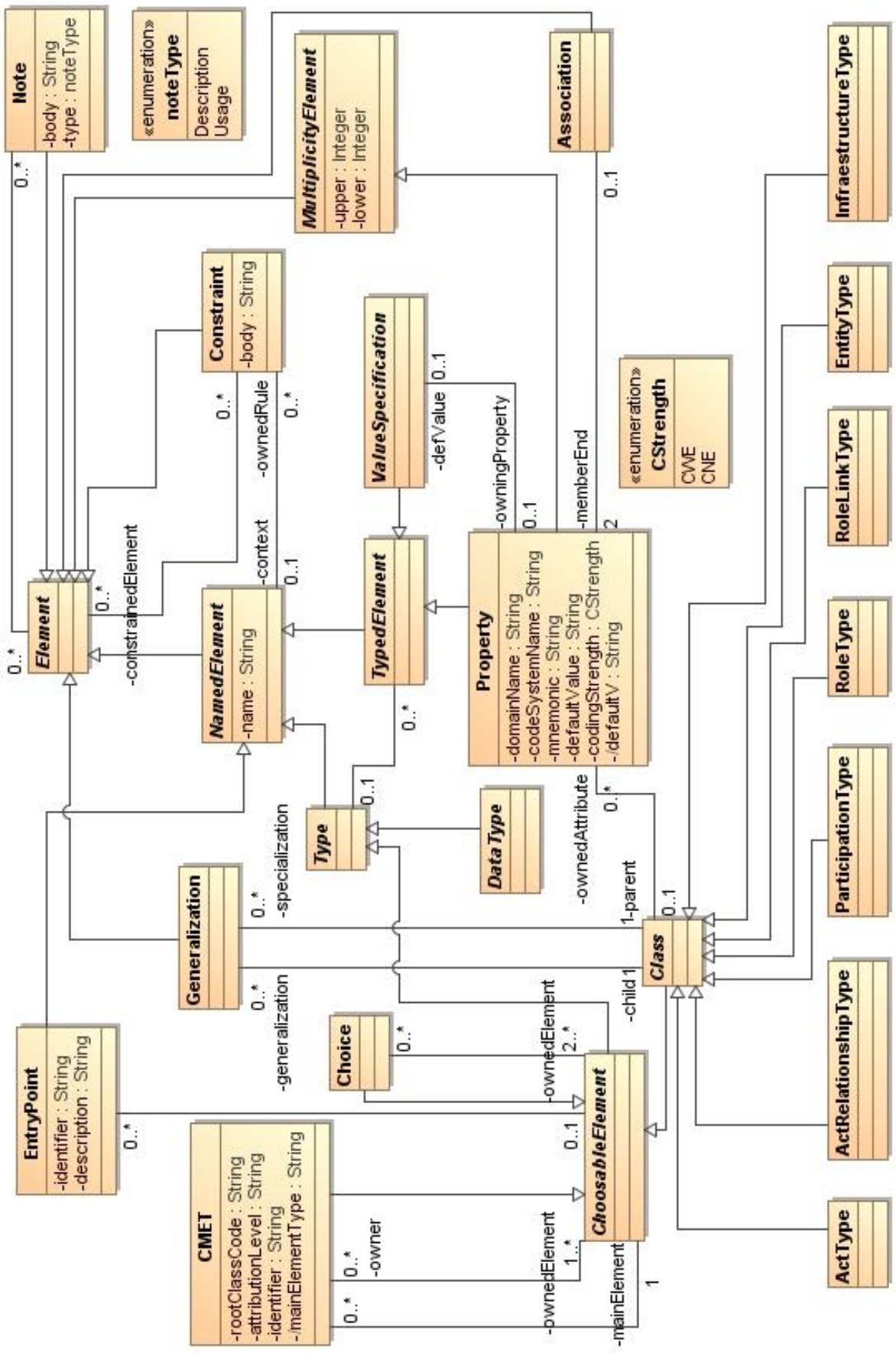


Figura 13.14 Metamodelo de HL7

13.2 Definición del metamodelo en USE

Como se afirmó anteriormente, uno de los motivos que nos llevaron a diseñar el metamodelo de HL7, es el poder darnos cuenta a partir de él, de si se habían definido correctamente todas las características de los elementos que componen el estándar HL7 y las relaciones que se dan entre ellos.

Para ello, se ha instanciado un subconjunto de esquemas conceptuales del estándar siguiendo las reglas definidas en el metamodelo que diseñamos. Con esto, se consigue de forma rápida encontrar posibles errores en él, puesto que por ejemplo, si intentamos instanciar una relación o un atributo que no existe en el metamodelo, el software que estemos usando nos avisará de que lo que intentamos hacer no es correcto según lo definido.

Para llevar a cabo esta tarea, se ha utilizado USE, una herramienta CASE (*Computer Aided Software Engineering*) que permite definir esquemas conceptuales en UML usando un lenguaje de modelado textual propio. La versión del software que se ha usado es la 2.6.2.

USE ofrece múltiples funcionalidades que resultan útiles a las personas que trabajan en el ámbito de la ingeniería del software. En este proyecto, no obstante, nos valdremos principalmente de una de ellas. USE permite realizar instancias de un esquema definido por nosotros mismos y validarlas contra ellos. De esta manera, si se intenta instanciar algo que no está definido en el esquema o que viola las restricciones de integridad definidas, el software advierte de ello. Esto último, tal y como se ha afirmado anteriormente, ha resultado ser de mucha ayuda de cara a detectar errores de diseño que se habían cometido.

En esta sección, se explica brevemente cómo se ha definido el metamodelo utilizando la herramienta USE. Sólo se expondrán aquellos elementos que resultan más relevantes y a partir de los cuales se puede llegar a entender la definición completa, cuyo código se encuentra en el anexo A. El lector que desee profundizar más sobre las características que proporciona este software puede visitar su página web, disponible en [Usew]

13.2.1 Definición de clases y asociaciones

Para definir una clase en USE, se utiliza la palabra clave *class*. Si se quiere indicar que ésta hereda de otra, se hace escribiendo el carácter '<' seguido del nombre de la clase padre. Por otro lado, los atributos se definen siguiendo la notación *nombre* : *tipo*, siendo necesario que la

lista se encuentre precedida por la palabra clave *attribute*. Por último, cuando el software se encuentra con la palabra *end*, interpreta que la definición de la clase ha terminado.

Así, la clase *EntryPoint* definida en el metamodelo de HL7, que hereda de *NamedElement* y contiene los atributos *identifier* y *description*, ambos de tipo *string*, se definiría tal y como se indica en el ejemplo 13.1.

```
class EntryPoint < NamedElement
attributes
  identifier: String
  description: String
end
```

Ejemplo 13.1 Definición de la clase Entry Point del metamodelo de HL7 en USE

Por otro lado, para definir una asociación, se utiliza la palabra clave *association* seguida de su nombre. Estas dos palabras, deben aparecen seguidas de la palabra clave *between*, que indica que a continuación se encuentran definidos sus dos extremos. Para definir cada uno de ellos, se utiliza la siguiente notación: *nombreClaseExtremo[multiplicidad] role nombreRol*, siendo el nombre de rol opcional, en caso que no se produzca ambigüedad. Para acabar, al igual que ocurre con la definición de clases, el fin de la definición se indica por medio de la palabra clave *end*.

De este modo, la asociación entre las clases *Class* y *Property*, que indica los atributos que pertenecen a una clase determinada, se define tal y como se encuentra indicado en el ejemplo 13.2.

```
association ClassToOwnedAttributeofProperty between
  Class[0..1]
  Property[*] role ownedAttribute
end
```

Ejemplo 13.2 Definición de la asociación entre Class y Property en USE

13.2.2 Definición de las restricciones en OCL

En la sección dedicada a la explicación del diseño del metamodelo de HL7, se han indicado las restricciones de integridad necesarias en lenguaje natural. En este apartado, se encuentran definidas formalmente en OCL. Para las instancias del metamodelo que definamos USE nos permite validar esas restricciones OCL, por lo que nos será de gran ayuda para saber si se han definido correctamente.

Las restricciones de integridad contenidas en el metamodelo de HL7 son las siguientes:

-"Si un miembro de la asociación es una clase *ActType*, el otro tiene que ser una clase *ActRelationshipType* o *ParticipationType*". (Ejemplo 13.3)

```
context Association inv
UnActSeRelacionaConUnaActRelationshipoConUnaParticipation:
(self.memberEnd->exists(me | me.type.oclIsTypeOf(ActType))) implies
(self.memberEnd->exists(me | me.type.oclIsTypeOf(ActRelationshipType)
or me.type.oclIsTypeOf(ParticipationType)or
(me.type.oclIsTypeOf(Choice) and
(me.type.oclAsType(Choice).ownedElement->forAll(oe |
oe.oclIsTypeOf(ActRelationshipType) or
oe.oclIsTypeOf(ParticipationType))))or (me.type.oclIsTypeOf(CMET) and
(me.type.oclAsType(CMET).mainElement.oclIsTypeOf(ActRelationshipType)
or
me.type.oclAsType(CMET).mainElement.oclIsTypeOf(ParticipationType)))
))
```

Ejemplo 13.3 Restricción de integridad del metamodelo de HL7 (I)

Existen diversas restricciones muy similares a la expuesta anteriormente, puesto que al igual que una clase de tipo *Act* sólo puede relacionarse con otras de determinado tipo, lo mismo ocurre con todas las demás. Recordemos por ejemplo, que una clase de tipo *Participation*, sólo puede estar relacionada con otras de tipo *Act* y *Role*. Debido a que todas esas reglas son muy similares entre sí, no se expondrán en este apartado, no obstante, se pueden consultar en el código completo del anexo A.

-"Un CMET no puede incluirse a sí mismo". (Ejemplo 13.4)

```
Context CMET inv CMETIncluyeSuElementoPrincipal:
  self.ownedElement->exists(oe | oe = self.mainElement)
```

Ejemplo 13.4 Restricción de integridad del metamodelo de HL7 (II)

-"El elemento principal de un CMET tiene que estar entre los elementos que éste contiene". (Ejemplo 13.5)

```
Context CMET inv CMETIncluyeSuElementoPrincipal:
  self.ownedElement->exists(oe | oe = self.mainElement)
```

Ejemplo 13.5 Restricción de integridad del metamodelo de HL7 (III)

-"Un *choice* no puede incluirse a sí mismo". (Ejemplo 13.6)

```
context Choice inv ChoiceNoSeIncluyeASiMismo:
  not self.ownedElement->exists(oe | oe = self)
```

Ejemplo 13.6 Restricción de integridad del metamodelo de HL7 (IV)

-"Un *choice* sólo puede contener elementos que sean compatibles entre sí". (Ejemplo 13.7)

```
context Choice inv ChoiceIncluyeElementosCompatibles:
  self.ownedElement->forAll(ce1,ce2:ChoosableElement | ce1.conformsTo(ce2))
```

Ejemplo 13.7 Restricción de integridad del metamodelo de HL7 (V)

Esta última, es la más complicada de definir. Está claro que todas las clases contenidas en un determinado *choice*, deben ser del mismo tipo, por ejemplo, si hay una que es de tipo *Act*, todas las demás deben ser también del tipo *Act*. No obstante, el hecho que un *choice* pueda contener otros *choices* y CMETs además de clases, complica la definición de esta restricción, obligándonos a decidir cuándo estos tres elementos son compatibles entre sí.

Así, se deben tener en cuenta las cuatro reglas siguientes para poder comprobar que todos los elementos de un *choice* son compatibles entre sí:

- Una clase es compatible con otra clase si ambas son del mismo tipo, un *choice* es compatible con otro *choice* si todos sus elementos son del mismo tipo, y un CMET es compatible con otro CMET si los elementos principales de ambos son del mismo tipo.
- Una clase es compatible con un CMET, si el elemento principal de este último, es del mismo tipo que el de la clase.
- Una clase es compatible con un *choice*, si todos los elementos contenidos en el *choice* son del mismo tipo que la clase.
- Un CMET, es compatible con un *choice*, si el tipo de todos los elementos de este último es igual que el del elemento principal del CMET en cuestión.

Todas estas comprobaciones, tal y como se puede apreciar en la restricción definida anteriormente, se realizan en una función llamada *conformsTo* que recibe dos elementos de tipo clase, *choice* o CMET, en definitiva de tipo *ChoosableElement* (ver metamodelo), y devuelve un booleano que retorna cierto si ambos son compatibles entre sí según las reglas definidas anteriormente, y falso en caso contrario.

Esta función, es recursiva debido a que al encontrarnos con un *choice*, debemos comprobar sus elementos internos. La función, está declarada dentro de la clase *ChoosableElement*, debido a que tanto las clases, como los *choices* y los CMETs, son subclases suyas. Se puede encontrar en el ejemplo 13.8.

```
conformsTo(ce:ChoosableElement):Boolean =
    if (self.oclIsTypeOf(CMET)) then
      self.oclAsType(CMET).mainElement.conformsTo(ce)
```

```
    else if (self.oclIsTypeOf(Choice) and not
     self.oclAsType(Choice).ownedElement->exists(oe | oe = self))
then
       self.oclAsType(Choice).ownedElement
       ->forAll(oe|oe.conformsTo(ce))
    else if (self.oclIsTypeOf(EntityType)) then
      ce.oclIsTypeOf(EntityType) or (ce.oclIsTypeOf(CMET) and
      ce.oclAsType(CMET).mainElement.conformsTo(self))
      or (ce.oclIsTypeOf(Choice) and (not
      ce.oclAsType(Choice).ownedElement->exists(oe | oe = ce)) and
      ce.oclAsType(Choice).ownedElement
      ->forAll(coe|coe.conformsTo(self)))
…
else false
…
end
```

Ejemplo 13.8 Función "conformsTo" utilizada en el metamodelo de HL7

En la restricción presentada, se han omitido fragmentos, puesto que cuando *self* es una clase, la comprobación sea su tipo *Entity*, *Act*… es análoga.

Cabe destacar, que en esta restricción es necesario protegernos contra la posibilidad de caer en un bucle infinito. Para ello, cada vez que se llama de forma recursiva a la función pasando como parámetro un *choice*, se debe comprobar que éste no se contenga a sí mismo.

13.3 Instanciaciones del metamodelo

Como se ha comentado anteriormente, utilizando USE, a partir del metamodelo de HL7 diseñado, es posible instanciar esquemas conceptuales del estándar HL7 y comprobar que cumplen con lo definido en el metamodelo.

En este proyecto, se han realizado instancias de dos esquemas del estándar pertenecientes al ballot de septiembre de 2009. Una del llamado *Clinical Document Architecture*, que se puede encontrar en [CDA], y otra del D-MIM del dominio de *Scheduling*, disponible en [Sched].

En esta sección, se explica cómo realizar instanciaciones en la herramienta USE y se ofrecen algunos detalles relevantes sobre las dos instanciaciones mencionadas en el párrafo anterior.

13.3.1 Instanciaciones en USE

La herramienta USE permite crear instanciaciones tanto a partir de su editor gráfico de esquemas UML, como a partir de comandos. Esta última forma, resulta mucho más rápida y cómoda una vez se conoce la sintaxis del lenguaje.

En este apartado, se explican algunos de los comandos que necesitamos utilizar para crear las instanciaciones mencionadas anteriormente. Se acompañan de ejemplos referentes a la del esquema *Clinical Document Architecture*, para hacer la explicación más clara.

Básicamente existen tres tipos de comandos, todos ellos con una sintaxis muy simple:

- Crear una instancia de cualquiera de las clases: !create *NombreInstanciaClase* : *TipoClase*.
- Insertar dos *properties* en los extremos de una asociación: !insert (*property1*, *property2*) into *NombreInstanciaAsociación*.
- Dar valor a un atributo: !set *NombreInstanciaClase*.*NombreAtributo* := *valor*.

De este modo, si por ejemplo se quisiese crear la asociación existente entre las dos clases mostradas en la figura 13.15, lo haríamos tal y como se indica en el ejemplo 13.9.

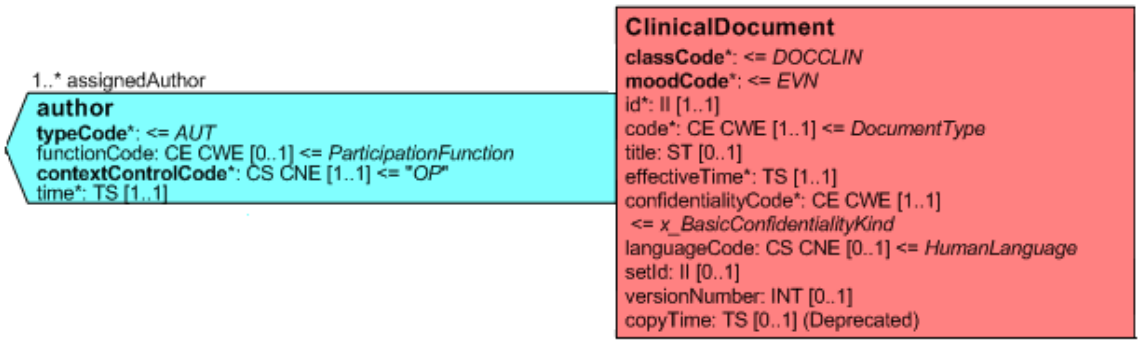


Figura 13.15 Asociación entre dos clases HL7, obtenida de [Hl7Ball]

Las instrucciones del ejemplo, se pueden dividir en la siguiente serie de pasos:

- Se crean las dos clases que se quieren relacionar y se les asigna un nombre (comandos 1, 2, 3 y 4).
- Se crea la asociación (comando 5).
- Se crean las dos *properties* de los extremos de la asociación (comandos 6 y 7).

- Se relacionan estas *properties* con sus tipos, es decir, con las clases creadas inicialmente (comandos 8 y 9).
- Se relacionan las dos *properties* con la asociación (comandos 10 y 11).
- Se da valor a las multiplicidades de las *properties* (comandos 12, 13, 14 y 15).

```
1.  !create ClinicalDocument : ActType
2.  !set ClinicalDocument.name := 'ClinicalDocument'
3.  !create Author : ParticipationType
4.  !set Author.name := 'Author'
5.  !create ClinicalDocumentAuthor : Association
6.  !create AuthorProperty1 : Property
7.  !create ClinicalDocumentProperty1 : Property
8.  !insert (ClinicalDocument,ClinicalDocumentProperty1) into
     TypeToTypedElement
9.  !insert (Author,AuthorProperty1) into TypeToTypedElement
10. !insert (AuthorProperty1,ClinicalDocumentAuthor) into
     memberEndOfPropertyToAssociation
11. !insert(ClinicalDocumentProperty1,ClinicalDocumentAuthor) into
     memberEndOfPropertyToAssociation
12. !set AuthorProperty1.upper := -1
13. !set AuthorProperty1.lower := 1
14. !set ClinicalDocumentProperty1.upper := 1
15. !set ClinicalDocumentProperty1.lower := 1
```

Ejemplo 13.9 Comandos USE para instanciar la asociación existente entre las clases "ClinicalDocument" y "Author"

En la figura 13.16, se encuentra la representación gráfica de la instanciación creada a partir de la lista de comandos mostrada arriba. Cabe destacar que, a excepción de las multiplicidades, todos los atributos de las instancias de *Property* quedan sin definir, puesto que sólo aplican cuando ésta actúa como atributo de una clase y no en todos los casos.

Además, es importante señalar que en las multiplicidades un "-1" equivale a "*", esto es debido a que en el metamodelo de UML, las multiplicidades son de tipo entero, y obviamente, "*" no lo es. Por ello, la solución que adoptan la mayoría de herramientas que trabajan con UML, consiste en representar "*" mediante el valor "-1".

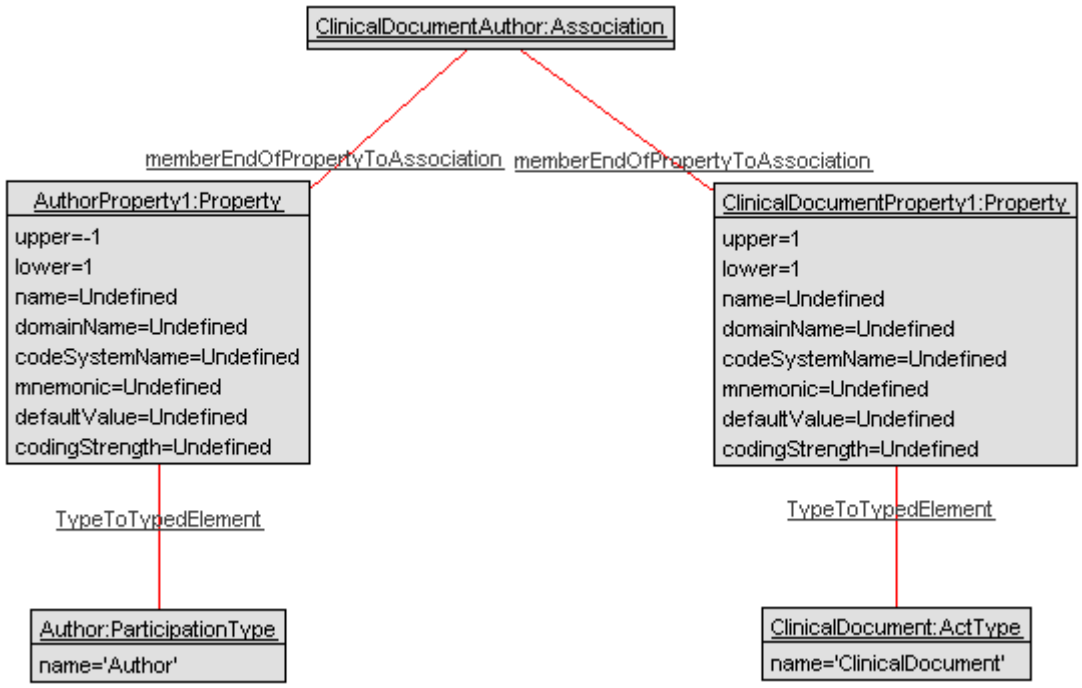


Figura 13.16 Instanciación de una asociación HL7 y sus dos extremos

En este ejemplo, se relaciona la clase *ClinicalDocument* de tipo *Act*, con la clase *Author* de tipo *Participation*. Para que este ejemplo acabe de cobrar sentido, es necesario señalar que en el archivo USE donde se ha definido el metamodelo, a la relación entre *Type* y *TypedElement* se la denomina *TypeToTypedElement* y a la existente entre *Property* y *Association*, *memberEndOfPropertyToAssociation*.

A partir del ejemplo anterior, se puede deducir cómo se instanciarían todos los demás elementos pertenecientes al metamodelo de HL7, pues todo se reduce a crear clases, relacionarlas y asignar valores a las *properties*. Así por ejemplo, para instanciar un *choice*, se crearía una clase de tipo *Choice*, sus miembros, y posteriormente, se crearía la asociación de la misma manera que en el ejemplo anterior. Además, habría que crear los atributos de las clases involucradas y asignarles valor a sus propiedades.

En el ejemplo 13.10, se muestra la instanciación del *choice* con nombre *AuthorChoice*, junto a la de los elementos que contiene. La información instanciada ha sido extraída del esquema *Clinical Document Architecture* citado anteriormente y se muestra en la figura 13.17. Por simplicidad, sólo se muestran los atributos de la clase *Person* y de cada uno, sólo se da valor al nombre y a la multiplicidad. En el ejemplo de la siguiente sección se da valor a todos los restantes.

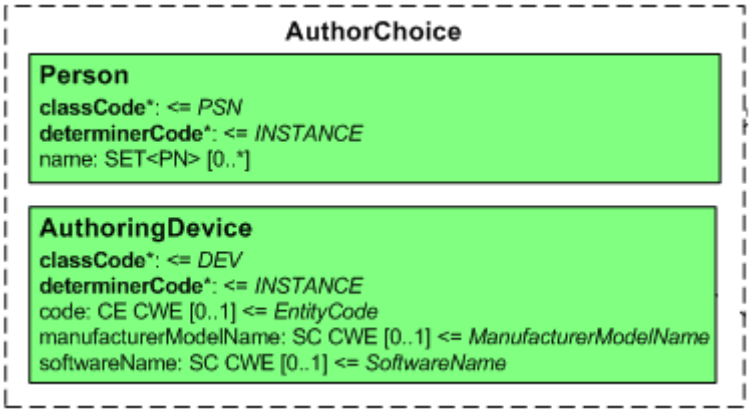


Figura 13.17 Choice AuthorChoice, obtenido de [HI7Ball]

```
1.  !create AuthorChoice : Choice
2.  !set AuthorChoice.name := 'AuthorChoice'
3.  !create Person : EntityType
4.  !set Person.name := 'Person'
5.  !create PersonAttribute1 : Property
6.  !set PersonAttribute1.name := 'classCode'
7.  !insert (Person, PersonAttribute1) into
     ClassToOwnedAttributeofProperty
8.  !create PersonAttribute2 : Property
9.  !set PersonAttribute2.name := 'determinerCode'
10. !insert (Person, PersonAttribute2) into
     ClassToOwnedAttributeofProperty
11. !create PersonAttribute3 : Property
12. !set PersonAttribute3.name := 'name'
13. !insert (Person, PersonAttribute3) into
     ClassToOwnedAttributeofProperty
14. !create AuthoringDevice : EntityType
15. !set AuthoringDevice.name := 'AuthoringDevice'
16. !insert (AuthorChoice,AuthoringDevice) into
     ChoiceToOwnedElementOfChoosableElement
17. !insert (AuthorChoice,Person) into
     ChoiceToOwnedElementOfChoosableElement
18. !set PersonAttribute1.upper := 1
19. !set PersonAttribute1.lower := 1
20. !set PersonAttribute2.upper := 1
21. !set PersonAttribute2.lower := 1
22. !set PersonAttribute3.upper := - 1
23. !set PersonAttribute3.lower := 0
```

Ejemplo 13.10 Comandos USE para instanciar el choice “AuthorChoice”

En la figura 13.18, se encuentra la representación gráfica de la instanciación creada a partir de la lista de comandos anterior.

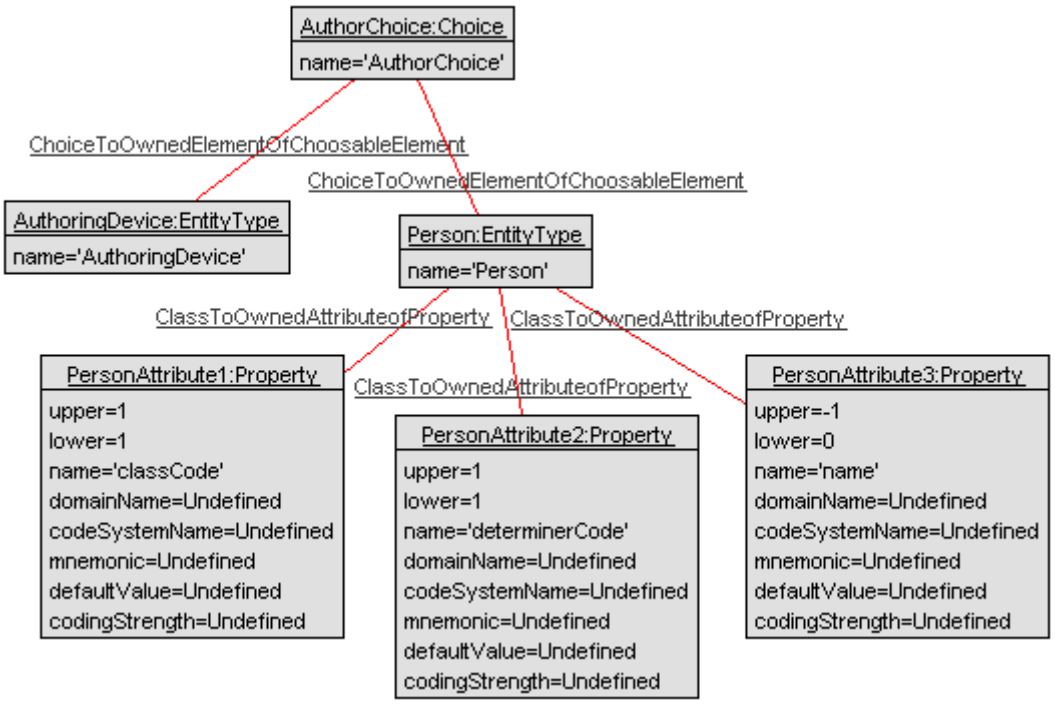


Figura 13.18 Instanciación del Choice AuthorChoice

13.3.2 Instanciación de Clinical Document Architecture

Se ha decidido instanciar el esquema *Clinical Document Architecture* (CDA), porque es uno de los que cuenta con un mayor porcentaje de uso entre las instituciones que deciden adoptar el estándar HL7. Además, resulta idóneo en nuestro caso, puesto que posee todos los elementos que aparecen en el metamodelo de HL7 diseñado, excepto los CMETs, que se tratan en la instanciación del siguiente apartado.

En este esquema conceptual, tal y como su nombre indica, se modela y se especifica tanto la estructura como la semántica de cualquier documento clínico, con el objetivo de que puedan ser intercambiados entre distintos sistemas de información que hayan adoptado el estándar HL7.

Los conceptos que presenta se encuentran divididos en dos secciones, la cabecera y el cuerpo. La primera, contiene la información más general como los datos del paciente, del doctor, etc., mientras que la segunda, contiene información específica del acto médico que representa el documento clínico. Así por ejemplo, si se tratase de un análisis de sangre, en el cuerpo del documento se encontrarían los valores referentes al número de leucocitos, hematíes, hemoglobina, etc.

Entre los conceptos que se encuentran modelados en dicho esquema, se pueden destacar algunos como los siguientes:

- Información del autor del documento y de la organización a la que pertenece.
- Detalles sobre el paciente al que hace referencia el documento: nombre, fecha de nacimiento, sexo, etc.
- Fecha de creación del documento.
- Elementos que indican las personas que participan en su redacción y las que deben ser informadas.
- Información específica sobre la actividad médica descrita en el documento que varía en función de si se trata de una intervención quirúrgica, observación, cita, análisis, medicación, etc.

Utilizando USE, es fácil comprobar que la instanciación realizada cumple con todas las reglas definidas en el metamodelo, incluyendo las restricciones de integridad. En la figura 13.19, se puede observar el entorno de trabajo proporcionado por la herramienta.

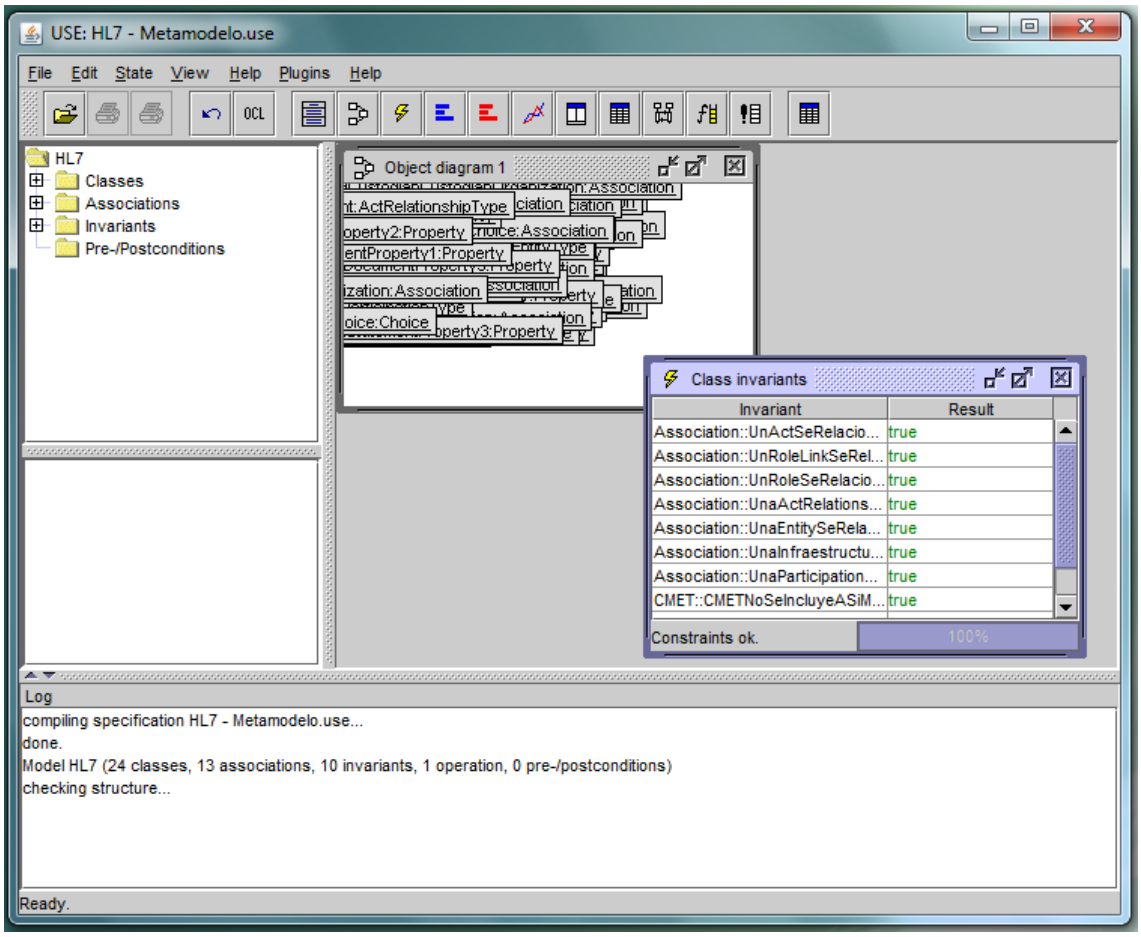


Figura 13.19 Captura de pantalla de la herramienta USE

Si se violase alguna restricción de integridad aparecería en la ventana llamada "*Class invariants*" y si hubiese alguna regla del metamodelo que no se cumpliese, saldría indicada en la ventana de "Log" tras el mensaje de "*checking structure*".

Tal y como se puede apreciar, este esquema contiene muchos elementos y obviamente, instanciar todos y cada uno de ellos, no sería de gran ayuda de cara a nuestros objetivos, puesto que la mayoría se instancian de forma análoga. Por este motivo, se ha instanciado sólo un subconjunto de él, pero procurando a la vez, que sea suficiente como para instanciar todos los distintos tipos de elementos y así, poder detectar cualquier error de diseño introducido en el metamodelo de HL7.

En este apartado, no se entra en detalle sobre esta instanciación, puesto que presentarla de forma gráfica resulta inviable debido a la gran cantidad de elementos que contiene. No obstante, se adjuntan en el anexo B todos los comandos utilizados para construirla, de forma que puedan ser consultados.

13.3.3 Instanciación del D-MIM del dominio de Scheduling

Con el objetivo de poder instanciar los CMETs, el único elemento que no aparece en el esquema *Clinical Document Architecture*, y de ser capaces de presentar una instanciación cuya representación gráfica pueda entenderse, se ha decidido instanciar un pequeño subconjunto del DMIM correspondiente al dominio de *Scheduling*, disponible en [Sched].

El subconjunto escogido es el que aparece en la figura 13.20. Los elementos que lo componen ya fueron expuestos en el capítulo 11, donde el esquema fue mostrado como ejemplo, por lo que no volverán a explicarse aquí. Además, por simplicidad, sólo se han instanciado algunos de los atributos, puesto que todos se instancian de forma análoga y son demasiados. Por la misma razón, no se han instanciado los tipos de datos.

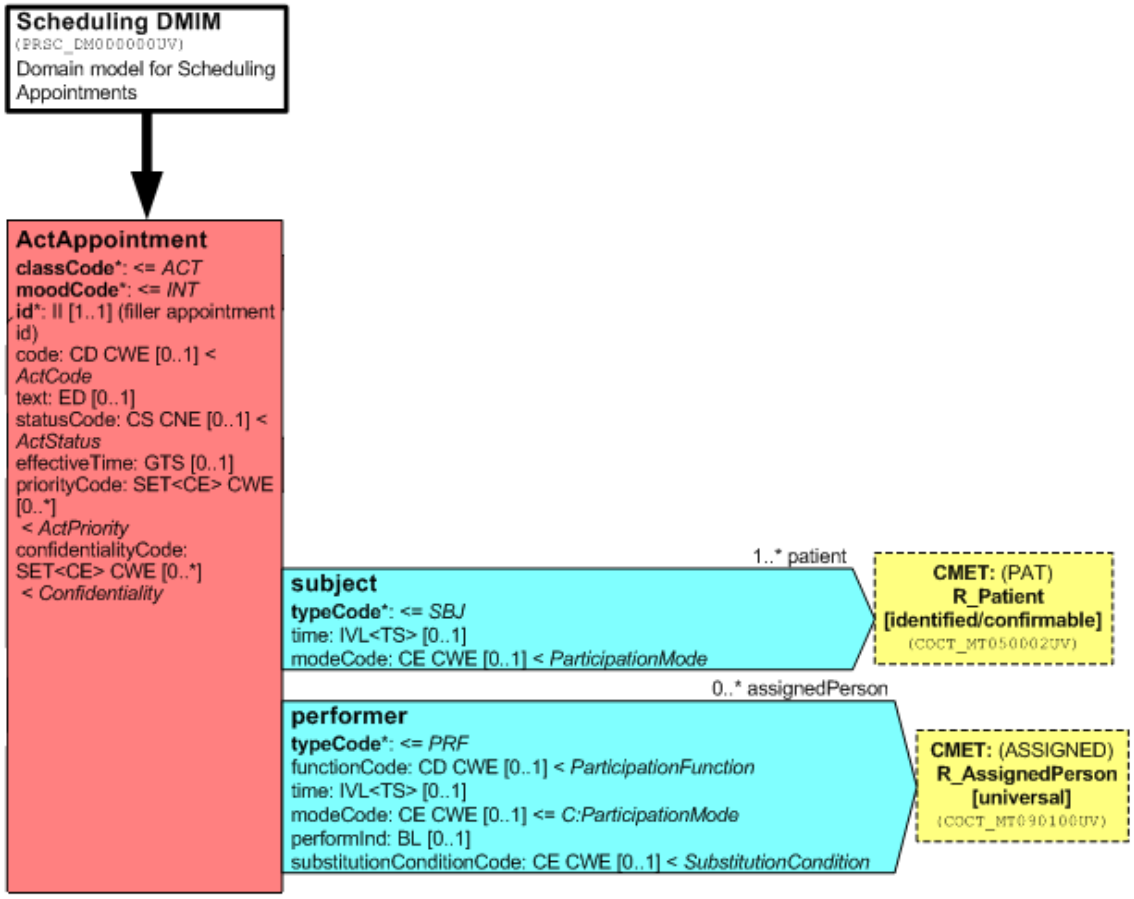


Figura 13.20 Subconjunto del D-MIM del dominio de Scheduling

La lista de comandos para construir la instanciación del fragmento mostrado puede encontrarse en el anexo C. Por otra parte, hemos creído conveniente en este caso, mostrar el resultado gráfico de ejecutar esa lista de comandos 13.21, debido a que al ser un esquema de tamaño relativamente reducido, se puede entender con facilidad.

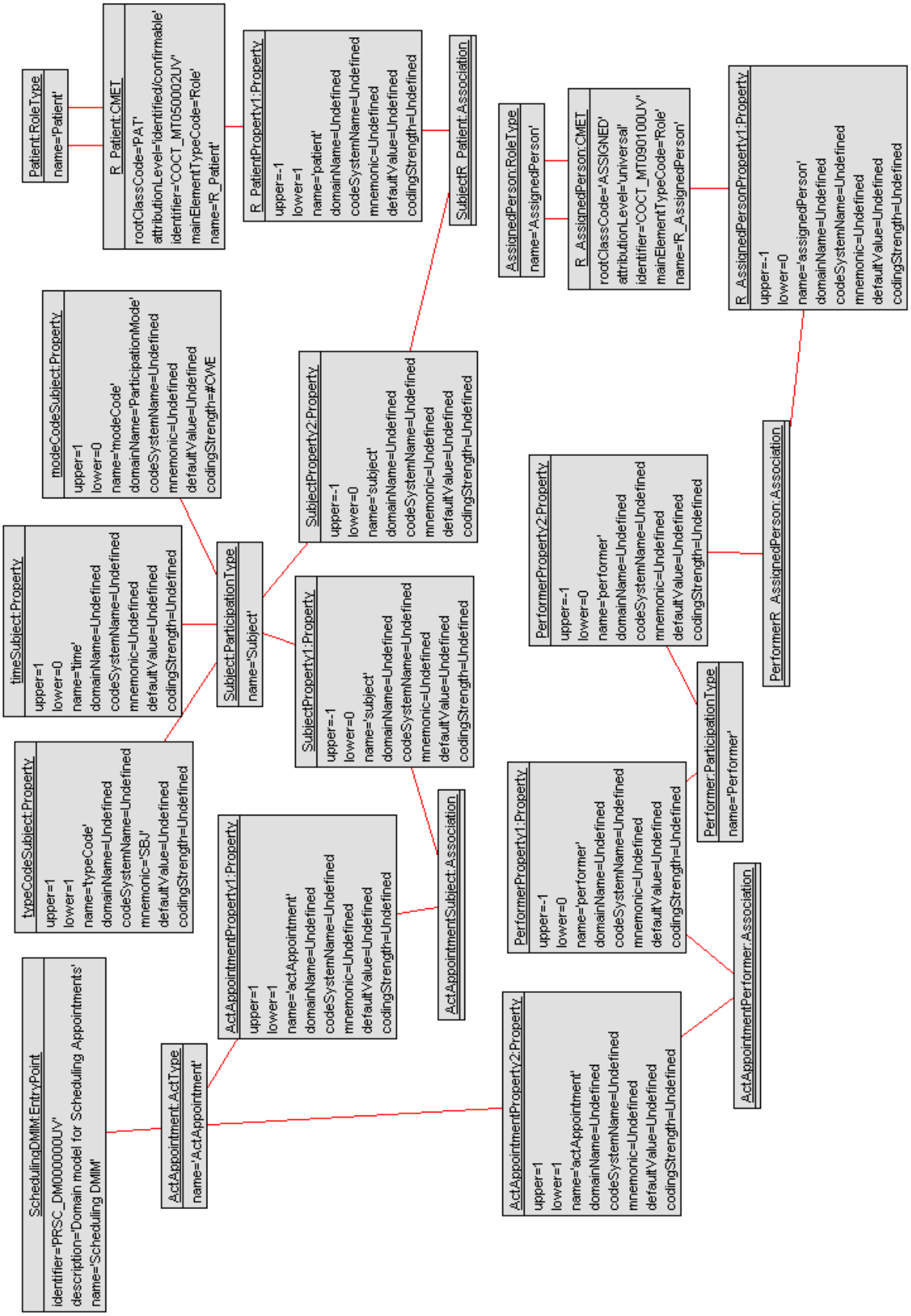


Figura 13.21 Instanciación de un subconjunto del D-MIM del dominio de Scheduling

Es fácil asociar cada uno de los elementos que aparecen en la instanciación a su análogo en el fragmento del esquema procedente del estándar HL7 si se tienen en cuenta un par de diferencias. La primera es que en la instanciación aparecen las clases principales del CMET y la segunda, que en el esquema las asociaciones se expresan simplemente dibujando las dos clases relacionadas juntas, mientras que en la instanciación en cambio, no se puede apreciar de forma tan directa puesto que se deben definir dos *properties* que actúen como extremos de la asociación.

13.4 Metamodelo HL7 en formato Ecore

Tal y como se comentó en la introducción de este capítulo, el metamodelo de HL7 diseñado, además de expresarse en UML y en la notación propia de USE, como se ha podido comprobar a lo largo de este capítulo, también se ha definido en formato *Ecore*.

Ecore es el formato propio de la herramienta Eclipse para la definición de metamodelos y el requerido por el software ATL para poder realizar las transformaciones. Pero además de servir en el proceso de transformación, también nos será de utilidad de cara a crear el *parser* de los archivos MIF. El cómo ayuda, se verá de forma detallada en los capítulos correspondientes a esas fases del proyecto.

El hecho de crear el archivo Ecore a partir de todo lo descrito en este capítulo, es una tarea muy sencilla, puesto que se trata simplemente de construir el metamodelo utilizando la herramienta gráfica de modelado de diagramas de la que dispone *Eclipse*, que aparece mostrada en la figura 13.22. Por este motivo, no se cree necesario entrar en más detalle en cuanto a este proceso.

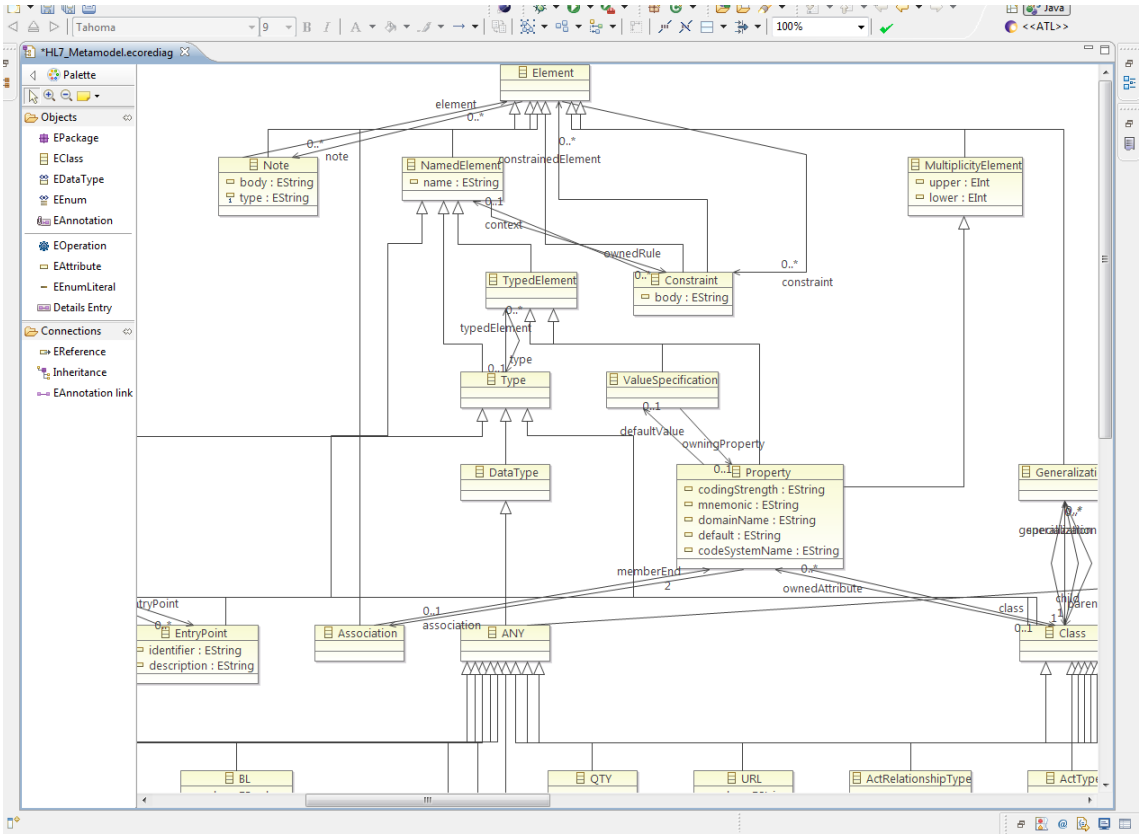


Figura 13.22 Captura de la herramienta gráfica de modelización de diagramas de Eclipse

13.5 Resumen

En este capítulo, se ha explicado la construcción del metamodelo de HL7 paso a paso y posteriormente, se han realizado instancias de modelos del estándar, con la ayuda de la herramienta USE, para cerciorarnos de que no había errores en el diseño.

Llegados a este punto, tenemos un metamodelo de HL7, que tal y como se verá en sus respectivos capítulos, resultará útil en las fases de construcción del *parser* de archivos MIF y en la del uso de la herramienta ATL.

14 PARSER DE FICHEROS MIF

En la fase del proyecto que se describe en este capítulo, el objetivo es desarrollar una herramienta que permita obtener, a partir de los esquemas conceptuales del estándar HL7, instancias XML del metamodelo diseñado en el capítulo anterior.

Obtener estas instancias XML resulta imprescindible de cara a poder ejecutar las transformaciones ATL. Recordemos que para convertir un esquema HL7, se necesitan los metamodelos de HL7 y UML, que llegados a este punto ya tenemos, pero también una instancia del metamodelo de HL7 escrita en XML, donde se exprese el conocimiento del esquema que se desea convertir.

Una de las decisiones principales que deben tomarse en este punto del proyecto, consiste en escoger de qué ficheros del estándar extraer el conocimiento que se expresa en los esquemas conceptuales.

En este capítulo, se exponen todas las posibilidades que se han valorado y se justifica la elección tomada. Por otro lado, también se explican algunos detalles sobre la implementación de la herramienta desarrollada para tratar esos ficheros, y finalmente, se ofrecen algunos ejemplos de instancias XML resultantes.

No obstante, antes de pasar a describir todos esos aspectos, se ha creído oportuno mostrar un ejemplo de uno de los ficheros XML resultantes, de cara a facilitar la comprensión de las siguientes secciones.

14.1 Ejemplo inicial

El esquema conceptual de HL7 escogido de cara a poder ilustrar los objetivos que deseamos cubrir en la fase del proyecto descrita en esta sección, es el *R_PatientClinical universal* con identificador *COCT_RM050004UV* y correspondiente a la versión normativa del estándar del año 2009 (figura 14.1). En este esquema, se representan los conceptos necesarios para identificar a un paciente.

Concretamente, se puede ver que el rol de paciente (*PatientClinical*), está relacionado con el ser vivo que juega ese rol (*E_LivingSubject*) y con el ámbito donde lo juega (*E_Organization*), que normalmente será un hospital. Finalmente, vemos que el rol de paciente participa como

sujeto de cualquier acto como observaciones, intervenciones, etc. La información relativa a esos tipos de actos se encuentra definida dentro del CMET *A_SupportingClinicalInformation*.

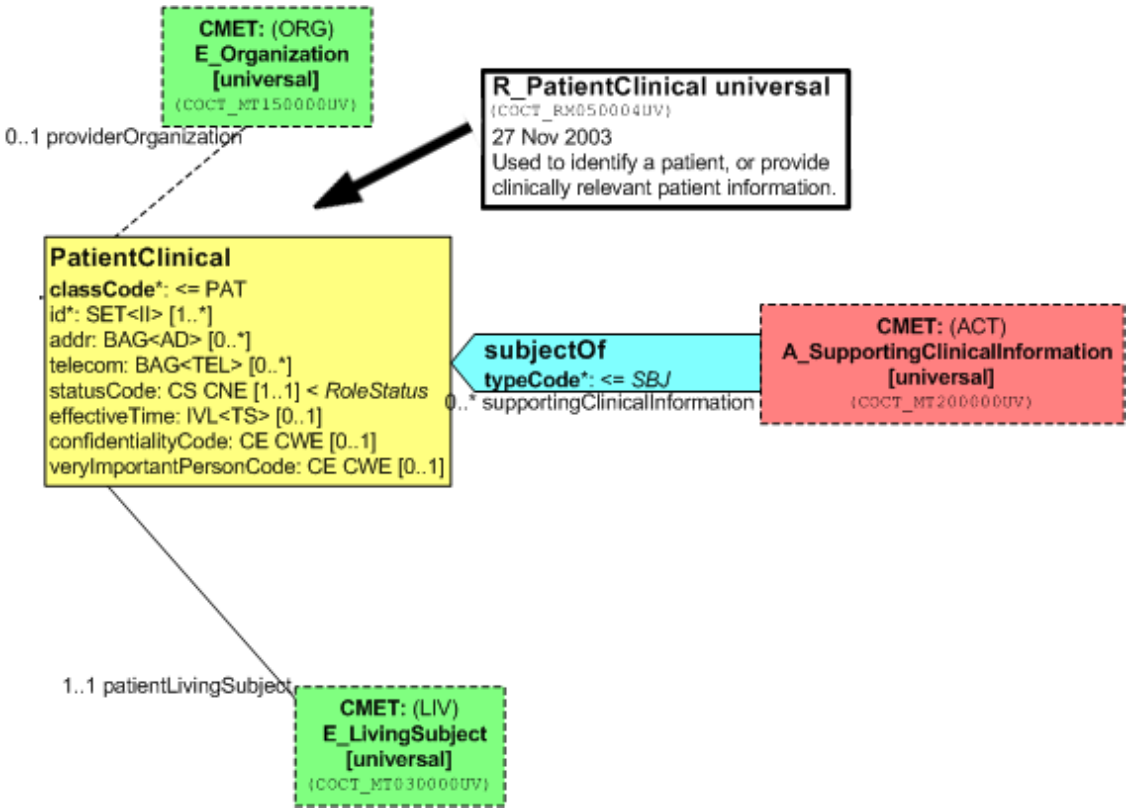


Figura 14.1 Esquema R_PatientClinical universal, obtenido de [HI7Ball]

En el ejemplo 14.1, se encuentra el código XML que corresponde al esquema anterior. Como se puede comprobar, a pesar de que el esquema conceptual seleccionado es uno de los más pequeños que se pueden encontrar en el estándar, el XML presenta un número de elementos considerable.

El objetivo que se persigue al mostrar este ejemplo, es simplemente introducir los resultados que se obtendrán al término del desarrollo de esta fase del proyecto, para que las explicaciones proporcionadas en este capítulo resulten más sencillas de entender. Por este motivo, por ahora, no se analiza en detalle este fichero XML, eso se hará en una sección posterior, aquí tan sólo se indica dónde se puede encontrar la información relativa a cada uno de los elementos del esquema del ejemplo dentro del código XML:

- La información de las dos clases que aparecen, se encuentra en los elementos 3 y 4.
- La de los CMETs en los elementos 5, 6 y 7.
- La del *Entry Point* en el 8.

- La de los atributos y sus tipos de datos entre los elementos 9 y 22 y en el 31.
- La de las asociaciones entre los elementos 23 a 35 exceptuando el 31.

```
1. <?xml version="1.0" encoding="ASCII"?>
2. <xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:hl7_metamodel="http://hl7_metamodel/1.0">
3.   <hl7_metamodel:RoleType name="PatientClinical" ownedAttribute="
      /7 /9 /11 /13 /14 /16 /18 /19"/>
4.   <hl7_metamodel:ParticipationType name="Subject"
      ownedAttribute="/28"/>
5.   <hl7_metamodel:CMET name="E_LivingSubjectUniversal"
      rootClassCode="LIV" attributionLevel="universal"
      identifier="COCT_MT030000UV04" mainClassType="Entity"/>
6.   <hl7_metamodel:CMET name="E_OrganizationUniversal"
      rootClassCode="ORG" attributionLevel="universal"
      identifier="COCT_MT150000UV02" mainClassType="Entity"/>
7.   <hl7_metamodel:CMET
      name="A_SupportingClinicalInformationUniversal"
      rootClassCode="ACT" attributionLevel="universal"
      identifier="COCT_MT200000UV01" mainClassType="Act"/>
8.   <hl7_metamodel:EntryPoint name="R_PatientClinical"
      identifier="COCT_MT050004UV01" description="27 Nov 2003 Used to
      identify a patient, or provide clinically relevant patient
      information." choosableElement="/0"/>
9.   <hl7_metamodel:CS name="CS"/>
10.  <hl7_metamodel:Property name="classCode" type="/6" upper="1"
      lower="1" codingStrength="CNE" mnemonic="PAT"
      codeSystemName="RoleClass"/>
11.  <hl7_metamodel:II name="II"/>
12.  <hl7_metamodel:Property name="id" type="/8" upper="-1"
      lower="1"/>
13.  <hl7_metamodel:AD name="AD"/>
14.  <hl7_metamodel:Property name="addr" type="/10" upper="-1"/>
15.  <hl7_metamodel:TEL name="TEL"/>
16.  <hl7_metamodel:Property name="telecom" type="/12" upper="-1"/>
17.  <hl7_metamodel:Property name="statusCode" type="/6" upper="1"
      lower="1" codingStrength="CNE" domainName="RoleStatus"/>
18.  <hl7_metamodel:IVL_TS name="IVL_TS"/>
19.  <hl7_metamodel:Property name="effectiveTime" type="/15"
      upper="1"/>
20.  <hl7_metamodel:CE name="CE"/>
21.  <hl7_metamodel:Property name="confidentialityCode" type="/17"
      upper="1" codingStrength="CWE" domainName="Confidentiality"/>
22.  <hl7_metamodel:Property name="veryImportantPersonCode"
      type="/17" upper="1" codingStrength="CWE"
      domainName="PatientImportance"/>
23.  <hl7_metamodel:Association memberEnd="/22 /21"/>
24.  <hl7_metamodel:Property name="patientEntityChoiceSubject"
      type="/2" upper="1" lower="1"/>
25.  <hl7_metamodel:Property name="playedRole" type="/0" upper="-1"/>
26.  <hl7_metamodel:Association memberEnd="/25 /24"/>
27.  <hl7_metamodel:Property name="providerOrganization" type="/3"
      upper="1"/>
28.  <hl7_metamodel:Property name="scopedRole" type="/0" upper="-1"/>
29.  <hl7_metamodel:Association memberEnd="/32 /27"/>
30.  <hl7_metamodel:Property name="subjectOf" type="/1" upper="-1"/>
31.  <hl7_metamodel:Property name="typeCode" type="/6" upper="1"
      lower="1" codingStrength="CNE" mnemonic="SBJ"
      codeSystemName="ParticipationType"/>
```

```
32.  <hl7_metamodel:Association memberEnd="/31 /30"/>
33.  <hl7_metamodel:Property name="actChoice" type="/4" upper="1"
      lower="1"/>
34.  <hl7_metamodel:Property name="participation" type="/1"
      upper="-1"/>
35.  <hl7_metamodel:Property name="role" type="/0" upper="1"
      lower="1"/>
36.</xmi:XMI>
```

Ejemplo 14.1 Fichero XML correspondiente al esquema R_PatientClinical universal

14.2 Archivos fuente del estándar

Como se ha afirmado anteriormente, escoger de entre los tipos de ficheros diferentes que contienen la información de los esquemas conceptuales del estándar, los adecuados, es una tarea delicada. Esto se debe a que existen diversos tipos de archivos y a que obviamente, la elección realizada influirá en el resto del proyecto y en sus resultados.

Básicamente, hay dos aspectos que se deben tener en cuenta al valorar cada uno de los distintos candidatos: la información que contiene y la dificultad a la hora de extraerla.

En cuanto al primero de ellos, hay que tener presente que no todos los tipos de ficheros contienen la misma información, algunos carecen de elementos que nos gustaría poder tratar, y otros constan de información extra que en nuestro caso no resulta de utilidad alguna. La carencia de información, supone un inconveniente más grande que el exceso, puesto que mientras que la falta de información no puede remediarse, el exceso simplemente dificulta el proceso de extracción de aquello que resulta de interés.

En cuanto a la dificultad de extraer la información existente en los distintos ficheros, no resulta un motivo de elección tan determinante como su carencia. Obviamente, es un aspecto a tener en cuenta, puesto que puede afectar de forma significativa al tiempo de desarrollo del *parser*, pero el resultado final del proyecto no depende de esta elección, a diferencia de lo que ocurre con la anterior.

Finalmente, cabe destacar que hay existen dos vías para obtener los ficheros del estándar que contienen el conocimiento expresado en sus esquemas conceptuales.

La primera es acudir a la sección de descargas de cualquiera de los *ballots* que se encuentran disponibles públicamente en internet. La segunda consiste en obtener la versión normativa de ese año, que sólo se encuentra disponible para miembros de la organización HL7.

Como ya se ha explicado anteriormente, los *ballots* se publican de forma cuatrimestral. En contraste, la versión normativa, se publica anualmente. Esto hace que la información que se encuentra en los ficheros de la versión normativa sea más estable y que normalmente, contenga menos errores. Estos dos últimos motivos, han resultado ser de suficiente peso como para decantarnos por esa versión en el proyecto que nos ocupa.

A continuación, se expone cada una de las posibles fuentes de información procesables de la versión normativa y se señalan las ventajas e inconvenientes que presentan.

14.2.1 MIF

El MIF (*Model Interchange Format*) es un formato de fichero con una estructura jerárquica en forma de árbol, similar a la de los XMLs. Este tipo de ficheros, contienen toda la información que se ha definido en el metamodelo de HL7 a excepción de las restricciones textuales.

Uno de los inconvenientes que presentan los ficheros MIFs, en nuestro caso, es el hecho que contienen información sobre el esquema gráfico al que hacen referencia. En este tipo de ficheros, podemos encontrar la posición que ocupa cada una de las clases representadas en el esquema gráfico o su tamaño. Esto hace que el tamaño del fichero y el número de nodos a tratar en su estructura arbórea se vea incrementado sustancialmente.

No obstante, al ser un tipo de fichero muy similar al XML, se pueden analizar utilizando funciones propias de la API de Java para tratar ficheros XML, llamada *Java API for XML*. Esto hace que el hecho que contengan información que no resulta útil en nuestro caso, no represente un gran problema, puesto que con esta API resulta muy sencillo hacer un filtrado de los nodos que se desean tratar.

En el ejemplo 14.2, se muestra un pequeño subconjunto de los elementos que se podrían encontrar en un fichero MIF. Tal y como se afirmó anteriormente, se puede comprobar que este tipo de archivos no son más que un XML adaptado, y que siguen su misma estructura basada en el uso de etiquetas con el objetivo de agrupar la información de los distintos elementos que componen el esquema.

En el fragmento mostrado, se incluye una parte de la información perteneciente al esquema conceptual con identificador *COCT_MT030007UV* de la versión normativa de 2009. Concretamente, la clase *Person* (figura 14.2). En ejemplo 14.2, se muestra la información relativa a esa clase y a un par de sus atributos (*classCode* y *determinerCode*). Como se puede

comprobar, el número de elementos necesarios para representar esa información en un fichero MIF, es bastante elevado.

Figura 14.2 Clase Person de HL7, obtenida de [Hl7Ball]

En este apartado, no se explica en detalle la estructura de este tipo de ficheros, no obstante, sí que se cree oportuno señalar brevemente qué representa cada uno de los elementos que se encuentran en el fichero:

- El elemento número 1, sirve para abrir la etiqueta de la clase *Person*, de este modo, todos los elementos contenidos en ella, representarán la información relativa a dicha clase.
- Los elementos 2 a 4, contienen información sobre la posición que ocupa la clase *Person* en la jerarquía de clases definida por el estándar.
- Los elementos 5 a 9, incluyen información sobre la versión gráfica del esquema conceptual.
- El resto de elementos, hacen referencia a los atributos *classCode* (11-17) y *determinerCode* (18-24). En el caso de *classCode*, por ejemplo, lo que resulta relevante son los atributos *name*, *maximumMultiplicity* y *minimumMultiplicity* del elemento 11 (*mif:attribute*), el elemento 15 (*mif:type*), que es el que contiene la información del tipo y finalmente, el elemento 16 (*mif:supplierDomainSpecification*), que contiene información adicional al ser el atributo de tipo código (*codingStrength*, *codeSystemName* y *mnemonic*).

```
1. <mif:class name="Person" isAbstract="false">
2. <mif:derivationSupplier staticModelDerivationId="1"
    className="Person"/>
3.   <mif:derivationSupplier staticModelDerivationId="2"
      className="Person"/>
4.   <mif:derivationSupplier staticModelDerivationId="3"
       className="Person"/>
5.   <mif:graphicRepresentation presentation="HL7">
6.     <mif:graphElement shapeId="Sheet.77"
        containerDiagramName="Main" shapeTemplate="Entity"
        isWidthAutoSize="false" isHeightAutoSize="true">
7.       <mif:position x="4.338" y="0.829"/>
8.       <mif:size width="2.232" height="3.048"/>
9.     </mif:graphElement>
10.  </mif:graphicRepresentation>
11.  <mif:attribute name="classCode" sortKey="1"
      minimumMultiplicity="1" maximumMultiplicity="1"
      conformance="R" isMandatory="true" isStructural="true">
12.    <mif:derivationSupplier staticModelDerivationId="1"
        className="Person" attributeName="classCode"/>
13.    <mif:derivationSupplier staticModelDerivationId="2"
        className="Person" attributeName="classCode"/>
14.    <mif:derivationSupplier staticModelDerivationId="3"
        className="Person" attributeName="classCode"/>
15.    <mif:type name="CS"/>
16.    <mif:supplierDomainSpecification codingStrength="CNE"
        codeSystemName="EntityClass" mnemonic="PSN"/>
17.  </mif:attribute>
18.  <mif:attribute name="determinerCode" sortKey="2"
      minimumMultiplicity="1" maximumMultiplicity="1"
      conformance="R" isMandatory="true" isStructural="true">
19.    <mif:derivationSupplier staticModelDerivationId="1"
        className="Person" attributeName="determinerCode"/>
20.    <mif:derivationSupplier staticModelDerivationId="2"
        className="Person" attributeName="determinerCode"/>
21.    <mif:derivationSupplier staticModelDerivationId="3"
        className="Person" attributeName="determinerCode"/>
22.    <mif:type name="CS"/>
23.    <mif:supplierDomainSpecification codingStrength="CNE"
        domainName="x_DeterminerInstanceKind"/>
24.  </mif:attribute>
       …
25.</mif:class>
```

Ejemplo 14.2 Fragmento del fichero MIF correspondiente al esquema con id COCT_MT030007UV

14.2.2 MIF-lite

Este tipo de ficheros contienen exactamente la misma información que los MIF, pero comprimida en una sola línea de texto. Esto obviamente, hace que el archivo pase a ser de menor tamaño y por tanto, pueda ser útil pensando en envíos a través de Internet.

No obstante, en nuestro caso particular, este hecho no representa ninguna ventaja, puesto que la diferencia de tiempo requerido entre procesar un fichero de tipo MIF y otro de tipo MIF-lite, utilizando la API de Java, resulta indistinguible en la práctica.

14.2.3 MIF2 y MIF2-lite

MIF2 es un tipo de fichero muy similar al MIF. En este tipo de fichero, tampoco aparecen las restricciones textuales y además, contienen algunos elementos menos que los MIF.

Por otro lado, la estructura del fichero es idéntica a la de los MIFs, con la salvedad de que la información gráfica aparece agrupada y separada del resto de elementos. En un principio, se podría pensar que esto supone una ventaja de cara a extraer la información, pero en la práctica, y de nuevo, gracias a la API de Java para tratar ficheros XML, esto no se traduce en un claro beneficio. De modo que, el hecho que este tipo de ficheros contenga menos información, resulta ser una circunstancia mucho más decisiva en nuestra elección.

14.2.4 XSD

Los archivos XSD del estándar, se usan como plantilla de los mensajes XML que al final se acaban transmitiendo entre distintos sistemas de información y sirven para validarlos, o dicho de otro modo, para asegurar que esos mensajes cumplen con lo definido en el estándar.

En este tipo de fichero, no se encuentra tanta información que no nos resulte de utilidad tal y como ocurría en el caso de los ficheros MIFs, no obstante, también es cierto que carecen de mucha de la información incluida en el metamodelo de HL7 diseñado. En concreto, en los ficheros XSDs, no se encuentra información acerca de los *Entry Points*, ni de las notas, ni tampoco de las restricciones de integridad. Este hecho, es un inconveniente demasiado grande y hace que la balanza se incline claramente a favor del MIF.

En el ejemplo 14.3, se muestra la información relativa a la misma clase *Person* utilizada en el ejemplo anterior. Se puede apreciar que, en un número similar de líneas, en el fichero XSD se expresa más información. Éste, se estructura de la siguiente manera:

- El primer elemento, indica que a continuación se proporciona la información relativa a la clase *Person*.
- Entre los elementos 4 y 21, se encuentran definidos los atributos pertenecientes a la clase.
- Entre los elementos 22 y 31, se ofrece la información relativa a las asociaciones de la clase.
- Entre los elementos 34 y 36, se pueden apreciar los atributos que determinan el tipo de la clase.

- El resto de elementos, básicamente, indican el principio de una sección.

```
1. <xs:complexType name="COCT_MT030007UV.Person">
2.   <xs:sequence>
3.     <xs:group ref="InfrastructureRootElements"/>
4.     <xs:element name="id" type="II" minOccurs="0"
        maxOccurs="unbounded"/>
5.     <xs:element name="name" type="EN" minOccurs="0"
        maxOccurs="unbounded"/>
6.     <xs:element name="desc" type="ED" minOccurs="0"
        maxOccurs="1"/>
7.     <xs:element name="statusCode" type="CS" minOccurs="0"
        maxOccurs="1"/>
8.     <xs:element name="administrativeGenderCode" type="CE"
        minOccurs="0" maxOccurs="1"/>
9.     <xs:element name="birthTime" type="TS" minOccurs="0"
        maxOccurs="1"/>
10.    <xs:element name="deceasedInd" type="BL" minOccurs="0"
        maxOccurs="1"/>
11.    <xs:element name="deceasedTime" type="TS" minOccurs="0"
        maxOccurs="1"/>
12.    <xs:element name="multipleBirthInd" type="BL"
        minOccurs="0" maxOccurs="1"/>
13.    <xs:element name="multipleBirthOrderNumber" type="INT"
        minOccurs="0" maxOccurs="1"/>
14.    <xs:element name="organDonorInd" type="BL" minOccurs="0"
        maxOccurs="1"/>
15.    <xs:element name="maritalStatusCode" type="CE"
        minOccurs="0" maxOccurs="1"/>
16.    <xs:element name="educationLevelCode" type="CE"
        minOccurs="0" maxOccurs="1"/>
17.    <xs:element name="disabilityCode" type="CE" minOccurs="0"
        maxOccurs="unbounded"/>
18.    <xs:element name="livingArrangementCode" type="CE"
        minOccurs="0" maxOccurs="1"/>
19.    <xs:element name="religiousAffiliationCode" type="CE"
        minOccurs="0" maxOccurs="1"/>
20.    <xs:element name="raceCode" type="CE" minOccurs="0"
        maxOccurs="unbounded"/>
21.    <xs:element name="ethnicGroupCode" type="CE" minOccurs="0"
        maxOccurs="unbounded"/>
22.    <xs:element name="asEmployment"
        type="COCT_MT030007UV.Employment" nillable="true"
        minOccurs="0" maxOccurs="unbounded"/>
23.    <xs:element name="asCitizen"
        type="COCT_MT030007UV.Citizen" nillable="true"
        minOccurs="0" maxOccurs="unbounded"/>
24.    <xs:element name="asStudent"
        type="COCT_MT030007UV.Student" nillable="true"
        minOccurs="0" maxOccurs="unbounded"/>
25.    <xs:element name="asMember" type="COCT_MT030007UV.Member"
        nillable="true" minOccurs="0" maxOccurs="unbounded"/>
26.    <xs:element name="asOtherIDs"
        type="COCT_MT030007UV.OtherIDs" nillable="true"
        minOccurs="0" maxOccurs="unbounded"/>
27.    <xs:element name="contactParty"
        type="COCT_MT030007UV.ContactParty" nillable="true"
        minOccurs="0" maxOccurs="unbounded"/>
28.    <xs:element name="guardian"
        type="COCT_MT030007UV.Guardian" nillable="true"
```

```
        minOccurs="0" maxOccurs="unbounded"/>
29.     <xs:element name="guarantor"
        type="COCT_MT030007UV.Guarantor" nillable="true"
        minOccurs="0" maxOccurs="unbounded"/>
30.     <xs:element name="birthPlace"
        type="COCT_MT030007UV.BirthPlace" nillable="true"
        minOccurs="0" maxOccurs="1"/>
31.     <xs:element name="languageCommunication"
        type="COCT_MT030007UV.LanguageCommunication"
        nillable="true" minOccurs="0" maxOccurs="unbounded"/>
32. </xs:sequence>
33. <xs:attributeGroup ref="InfrastructureRootAttributes"/>
34. <xs:attribute name="nullFlavor" type="NullFlavor"
     use="optional"/>
35. <xs:attribute name="classCode" type="EntityClass"
     use="required" fixed="PSN"/>
36. <xs:attribute name="determinerCode"
     type="x_DeterminerInstanceKind" use="required"/>
37.</xs:complexType>
```

Ejemplo 14.3 Fragmento del fichero XSD correspondiente al esquema con id COCT_MT030007UV

14.2.5 HTML

En el estándar, se pueden encontrar ficheros HTML que contienen información acerca de los esquemas, estructurada en un elemento de tipo tabla. No obstante, estos ficheros no representan información sobre los D-MIM (*Domain Message Information Model*) o los R-MIM (*Refined Message Information Model*), sino que contienen la información relativa a los HMD (*Hierarchical Message Descriptions*), a diferencia de los tipos de ficheros analizados anteriormente. La diferencia entre el contenido de los R-MIM y los HMD, es prácticamente inapreciable en la mayoría de los casos, no obstante, en algunos otros no, ya que existen varios HMD distintos entre sí, que derivan del mismo R-MIM. Este hecho representa un problema en nuestro caso, puesto que como ya se explicó, el nivel más bajo que deseamos tratar en este proyecto es el de los R-MIM.

De todas maneras, aunque representasen la información de los R-MIM, no constituirían la mejor opción para nosotros, puesto que al igual que ocurre en los MIF, la información de las restricciones textuales no se encuentra disponible. Además, tampoco se puede encontrar en ellos la información relativa a los *Entry Points*.

El fragmento de la tabla HTML que representa la información de la clase *Person* citada anteriormente, se encuentra en la figura 14.3. Las casillas blancas contienen la información de sus atributos, mientras que las que están coloreadas con un tono grisáceo, indican las asociaciones que tiene con otras clases.

Person
classCode [1..1] (M) Person (CS) {CNE:C:**EntityClass**:**PSN**}
determinerCode [1..1] (M) Person (CS) {CNE:V:**x_DeterminerInstanceKind**}
id [0..*] Person (SET<II>)
name [0..*] Person (BAG<EN>)
desc [0..1] Person (ED)
statusCode [0..1] Person (CS) {CNE:D:**EntityStatus**}
administrativeGenderCode [0..1] Person (CE) {CWE:D:**AdministrativeGender**}
birthTime [0..1] Person (TS)
deceasedInd [0..1] Person (BL)
deceasedTime [0..1] Person (TS)
multipleBirthInd [0..1] Person (BL)
multipleBirthOrderNumber [0..1] Person (INT)
organDonorInd [0..1] Person (BL)
maritalStatusCode [0..1] Person (CE) {CWE:D:**MaritalStatus**}
educationLevelCode [0..1] Person (CE) {CWE:D:**EducationLevel**}
disabilityCode [0..*] Person (SET<CE>) {CWE:D:**PersonDisabilityType**}
livingArrangementCode [0..1] Person (CE) {CWE:D:**LivingArrangement**}
religiousAffiliationCode [0..1] Person (CE) {CWE:D:**ReligiousAffiliation**}
raceCode [0..*] Person (SET<CE>) {CWE:D:**Race**}
ethnicGroupCode [0..*] Person (SET<CE>) {CWE:D:**Ethnicity**}
asEmployment [0..*] (Employment)
asCitizen [0..*] (Citizen)
asStudent [0..*] (Student)
languageCommunication [0..*] (LanguageCommunication)
Inherits associations from EntityChoiceSubject

Figura 14.3 Tabla HTML que representa la clase Person de HL7, obtenida de [Hl7Ball]

14.2.6 Decisión

Tal y como se puede deducir a partir de las descripciones de cada uno de los tipos de fichero expuestos, el formato que mejor se adapta a las necesidades del presente proyecto es el MIF. El único inconveniente que presenta, es que no podemos extraer de él la información relativa a las restricciones textuales existentes en los esquemas conceptuales, no obstante, también es cierto que ninguno de los otros formatos incorpora esta información. Como se ha podido comprobar, las demás opciones, o bien presentan además otras carencias de información, o bien extraer su información resulta menos cómodo.

En el proyecto que nos ocupa, el hecho que las restricciones textuales no estén disponibles en ningún formato procesable, provocará que aquellas que aparecen en los esquemas conceptuales, no aparezcan de ninguna manera en los ficheros UML resultantes, debido a que, lógicamente, teniendo en cuenta la gran cantidad de restricciones textuales que hay repartidas por todos los esquemas, resulta inasumible tratarlas todas sin poder valernos de ningún tipo de proceso automático. Este problema se trata con más detalle en la sección 18.2 del capítulo dedicado a los inconvenientes encontrados en el estándar.

La elección de los archivos fuente del estándar es importante, puesto que condiciona en gran medida la información contenida en los ficheros UML resultantes del proceso de transformación. Por suerte, en los últimos compases de este proyecto, tuvimos la oportunidad de organizar una reunión con Diego Kaminker, presidente de la delegación de HL7 en Argentina, y él ratificó la decisión de utilizar los ficheros MIF. Además, también afirmó que es más probable encontrar errores en los ficheros MIF de los *ballots* que en los de la versión normativa. Este hecho nos permite disipar todas las dudas en cuanto a la decisión tomada en esta sección.

14.3 Estructura de los ficheros MIF

En esta sección, se explica la estructura de los ficheros MIF, obviando todos aquellos aspectos que no resultan relevantes en nuestro caso, como la información gráfica.

Tal y como se ha afirmado anteriormente, los ficheros MIF siguen una estructura jerárquica en forma de árbol similar a la de los XML. Presentar todos los elementos en un solo árbol, podría resultar confuso, puesto que la cantidad de distintos elementos existentes es bastante elevada. Por este motivo, se ha decidido ir explicando su estructura por partes.

En cuanto a la presentación, con el objetivo de facilitar la comprensión, se muestran de color rojo los nodos simples, mientras que los expandibles cuya información se encuentra en los subnodos, se muestran de color azul. Por otro lado, los atributos que son relevantes en el proyecto que nos ocupa, se indican en el interior de los nodos a los que pertenecen.

14.3.1 Primeros niveles

El primer nodo, es siempre de tipo *serializedStaticModel*, y de él, cuelga la información del *Entry Point*, las clases, los *choices* y los CMETs del esquema representado (figura 14.4).

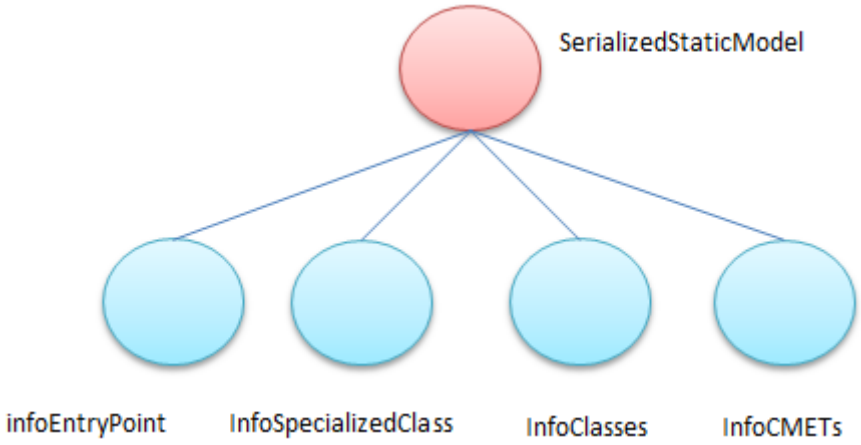


Figura 14.4 Nodo SerializedStaticModel de los ficheros MIF

14.3.2 Notas

Pese a no ser de los elementos más importantes, se cree oportuno comenzar mostrando la estructura relativa a las notas, puesto que la mayoría de los demás elementos pueden presentar notas, y por tanto, en su estructura referenciaran al fragmento mostrado en la figura 14.5.

En los nodos de tipo *description*, se encuentran las notas descriptivas, mientras que en los de tipo *usageNotes*, se encuentran las de uso. El texto de la nota en sí, se encuentra en el atributo *p* de los nodos de tipo *text*, hijos de los dos anteriores.

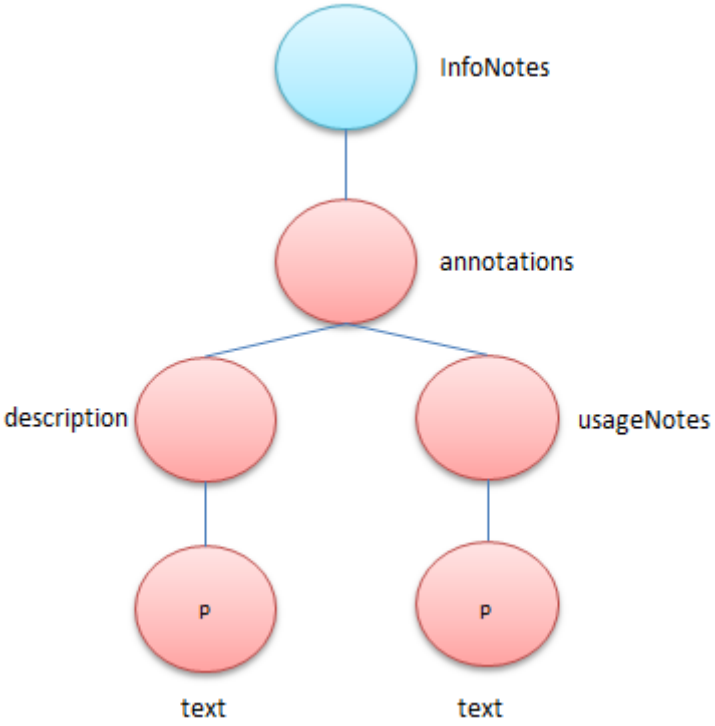


Figura 14.5 Nodo infoNotes de los ficheros MIF

14.3.3 Entry Point

La estructura de los *Entry Points* (14.6), es muy simple, puesto que el nodo que los representa sólo cuenta con un atributo, *name*, y con un hijo, que contiene la información de las notas que hacen referencia a ese *Entry Point* concreto.

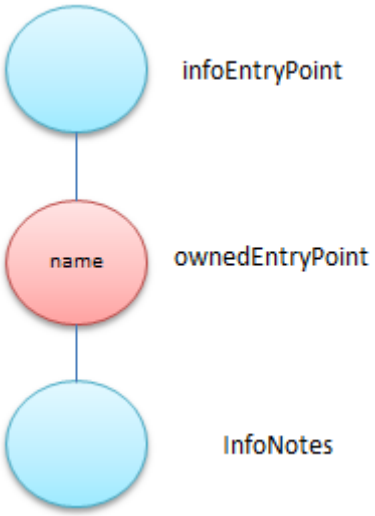


Figura 14.6 Nodo infoEntryPoint de los ficheros MIF

14.3.4 Clases y Choices

La información de las clases, así como la de los *choices*, aparece representada en nodos de tipo *class*. Este tipo, tal y como se puede apreciar en la figura 14.7, tiene en sus hijos nodos la información relativa a las notas que los afectan, los atributos que contienen (sólo en el caso de las clases), las asociaciones que definen, y los hijos especializados que contienen (sólo en el caso de los *choices*). En cuanto a sus atributos, el único que nos interesa de los nodos de tipo *class* es el que nos indica su nombre (*name*).

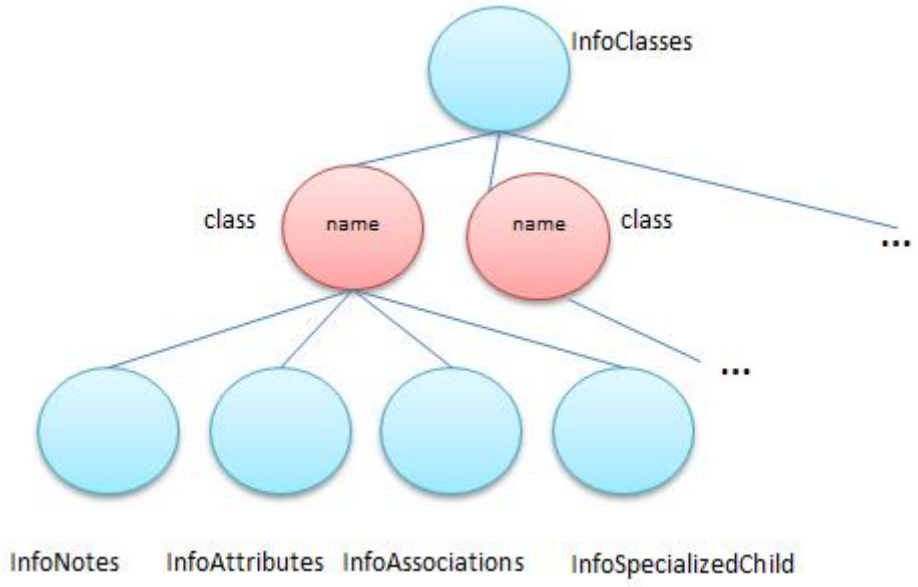


Figura 14.7 Nodo infoClasses de los ficheros MIF

14.3.5 Atributos

En los nodos de tipo *attribute,* se encuentran varias propiedades que nos interesa extraer: el nombre del atributo (*name*), su valor por defecto (*defaultValue*) y sus multiplicidades (*maxMult* y *minMult*).

Por otro lado, este tipo de nodos, presentan cuatro tipos de hijos distintos:

- *infoNotes*, que contiene la información relativa a las notas que le hacen referencia.
- *type*, que indica el tipo del atributo. Este nodo puede, de forma opcional, presentar otro nodo de tipo *supplierBindingArgumentDT*, que indica el subtipo. Así por ejemplo, si el tipo de dato fuese SET<INT>, en el nodo *type*, el atributo *name* tomaría el valor *SET* y en el nodo *supplierBindingArgumentDT*, tomaría el valor *INT*. En HL7, es posible encadenar varios niveles de tipos, dando como resultado SET<SET<INT>>, por ejemplo.

Por este motivo, los nodos de tipos *supplierBindingArgumentDT*, también pueden tener como hijos nodos de tipo *type*.

- businessName, que simplemente contiene un atributo *name* que representa el comentario que aparece escrito entre paréntesis al lado de algunos atributos en los esquemas conceptuales.
- supplierDomainSpecification, de donde se puede extraer la información relativa al dominio de vocabulario al que pertenece el atributo. Concretamente, en él se encuentran los atributos *codingStrength*, *mnemonic*, *codeSystemName* y *domainName*.

La estructura asociada a este tipo de nodo se muestra en la figura 14.8.

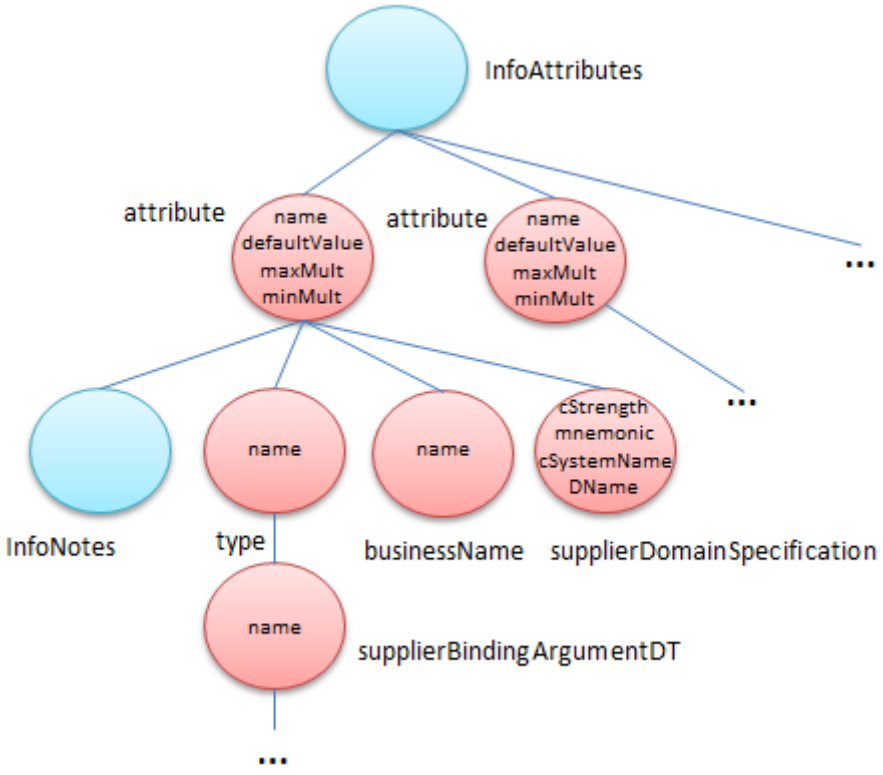


Figura 14.8 Nodo InfoAttributes de los ficheros MIF

14.3.6 Asociaciones

Por cada asociación en la que participa una clase determinada, existe un nodo de tipo *association*. Cada uno de estos, presenta siempre dos nodos hijos, *targetConnection* y *sourceConnection*, que contienen la información perteneciente a las dos *properties* presentes en los extremos de dicha asociación.

En los nodos de tipo *targetConnection*, se encuentran el nombre de rol y las multiplicidades de una de las *properties*. En cuanto a sus hijos, será un nodo de tipo *participantClass* si el extremo de la asociación es una clase o *choice* y de tipo *commonModelElementRef* si es un CMET. Es posible que en lugar de uno de estos elementos, exista una referencia a ellos, en este caso, el nodo será de tipo *reference*. Este último tipo de nodo, contiene un atributo *name*, que nos indica a cuál de los elementos definidos en el fichero MIF se está referenciando.

Por otra parte, en los nodos de tipo *sourceConnection*, encontramos un nodo hijo de tipo *nonTraversableConnection*, que no contiene ninguna información que resulte de interés en nuestro caso concreto, y en un nivel más bajo, un nodo de tipo *derivationSupplier*, que contiene el nombre de rol de la segunda *property* de la asociación.

El nodo *association* y la jerarquía que define se pueden encontrar en la figura 14.9.

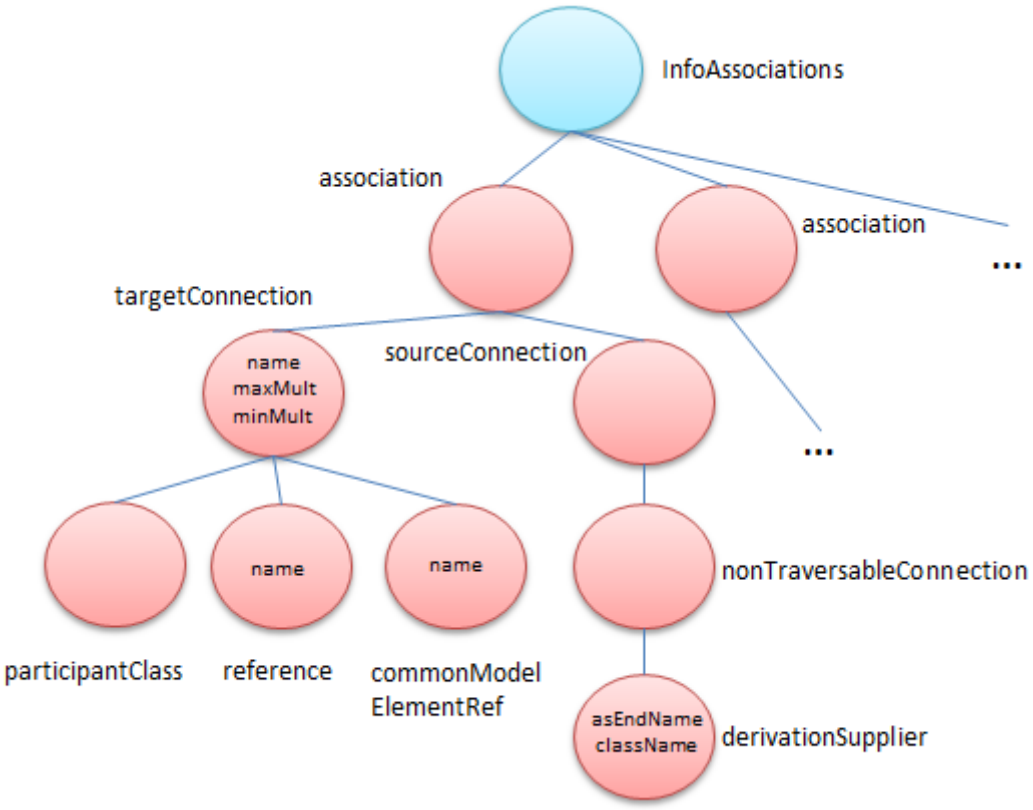


Figura 14.9 Nodo infoAssociations de los ficheros MIF

14.3.7 Clases especializadas

El tipo de nodo *specializedClass*, puede hacer referencia a la clase apuntada por el *Entry Point* del esquema o a miembros de *choices*. En sus nodos hijos, solamente se encuentra el tipo de nodo *class*, que ya se ha explicado anteriormente.

El fragmento de la estructura de los ficheros MIF relativo al tipo de nodo *specializedClass* puede encontrarse en la figura 14.10.

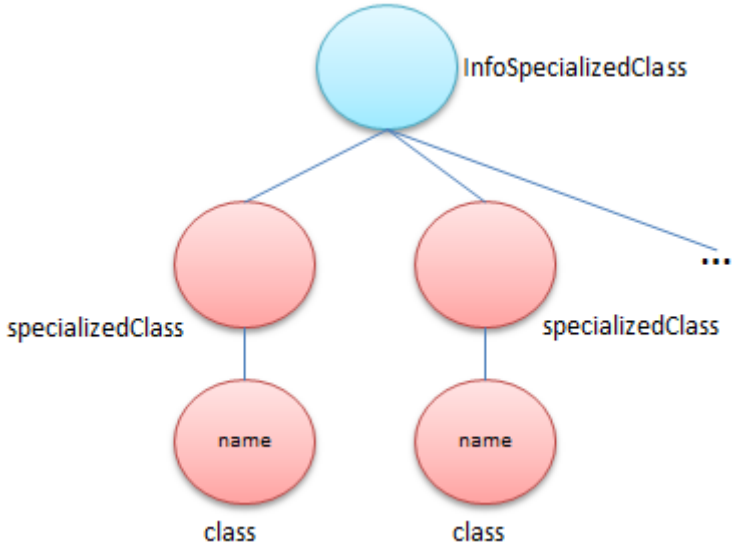


Figura 14.10 Nodo infoSpecializedClass de los ficheros MIF

14.3.8 Hijos especializados

El tipo de nodo *specializationChild*, se puede encontrar en los hijos de un nodo de tipo *class*, en caso que este último contenga la información referente a un *choice* y no a una clase, tal y como se explicó anteriormente.

Este tipo de nodo, puede tener como hijos a un nodo de tipo *specializedClass*, ya detallado anteriormente, o a una referencia. La estructura explicada se encuentra en la figura 14.11.

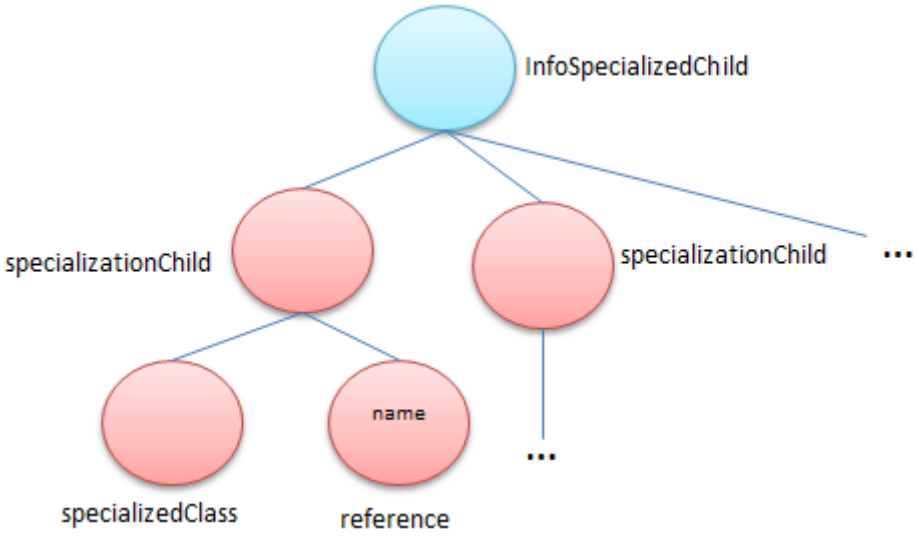


Figura 14.11 Nodo infoSpecializedChild de los ficheros MIF

14.3.9 CMETs

Los CMETs están representados por los nodos de tipo *commonModelElementRef*, que únicamente contienen un atributo *name*. En cuanto a sus nodos hijos, se encuentran *supplierStructuralDomain*, que contiene el *rootClassCode* del CMET en el atributo *mnemonic* y *generalizationParent*, que contiene su identificador en el atributo *name*. En la figura 14.12, se muestra la estructura descrita.

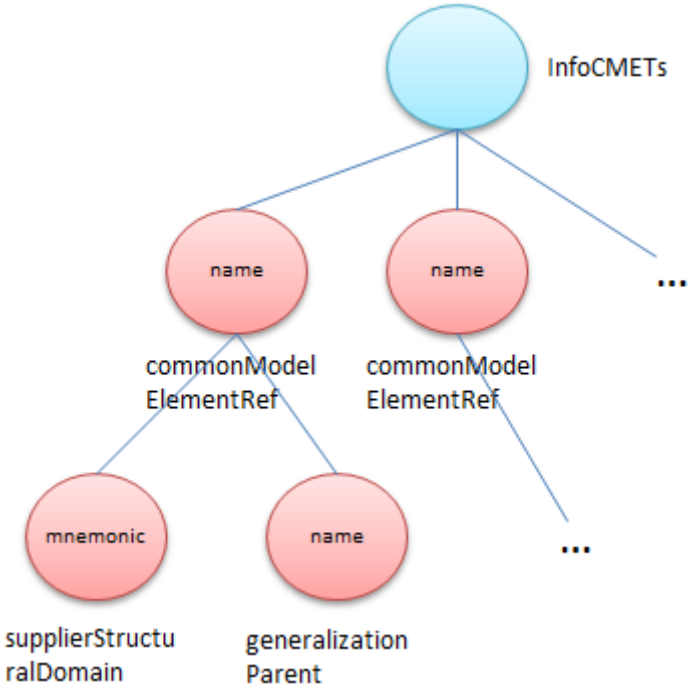


Figura 14.12 Nodo infoCMETs de los ficheros MIF

14.4 Desarrollo del parser de ficheros MIF

Tras haber escogido el tipo de ficheros desde los cuales extraer la información y haber analizado su estructura, el siguiente paso consiste en desarrollar una herramienta que a partir de esos ficheros fuente, sea capaz de generar archivos XML que sean instancias del metamodelo de HL7 diseñado en el capítulo anterior.

Dicha herramienta, se ha codificado en Java y básicamente, lo que hace es ir leyendo la estructura arbórea de los ficheros MIF de forma recursiva, con el objetivo de detectar todos los elementos definidos en el metamodelo de HL7, para posteriormente exportarlos a un fichero XML.

La forma más intuitiva para realizar esas tareas, requiere definir una clase Java para cada una de las metaclases que se pueden encontrar en el metamodelo de HL7. Así por ejemplo, si se define una clase CMET con los atributos que incluye en el metamodelo, al leer del fichero MIF la información relativa a un CMET, se podrá crear un nuevo objeto de tipo CMET, y de forma simple, se podrá asignar valores a sus atributos y relaciones a partir de métodos *setters* y *getters* que se hayan definido.

14.4.1 Generación de una API a partir del metamodelo de HL7

La idea anterior de declarar una clase con todos sus atributos junto a sus métodos *setters* y *getters* por cada una de las que existen en el metamodelo HL7, constituye una tarea tediosa y propensa a errores.

Por suerte para nosotros, el haber definido un metamodelo en formato *Ecore*, tal y cómo ya se afirmó anteriormente en este documento, nos resulta muy útil llegados a este punto, ya que *Eclipse* es capaz de generar una API que contiene todas esas clases y métodos que necesitamos, de forma automática.

Para ello, lo primero que se debe hacer es crear un nuevo fichero de tipo "*EMF Generator Model*" a partir del asistente proporcionado por *Eclipse* (figura 14.13). Una vez se tiene ese fichero, lo único que se debe hacer es buscar la opción llamada "*Generate Model Code*" (figura 14.14) y automáticamente se generarán todas las clases anteriormente descritas junto a sus métodos.

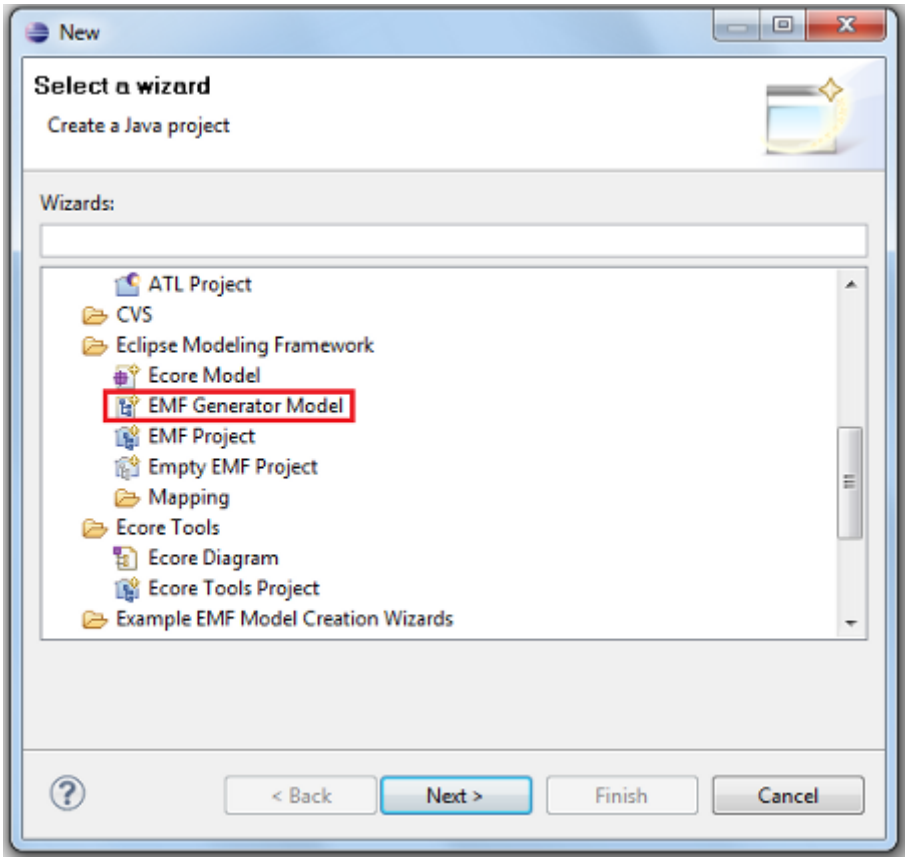


Figura 14.13 Captura del asistente de creación de nuevo proyecto de Eclipse

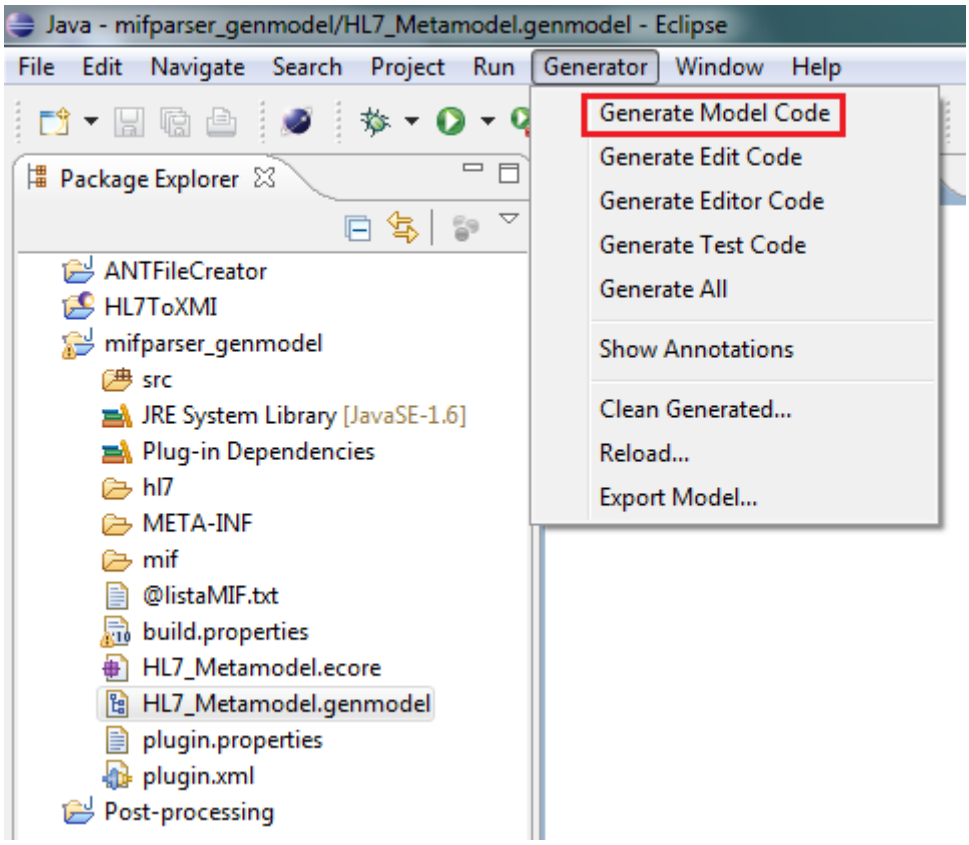


Figura 14.14 Captura de la opción de generar el código del modelo en Eclipse

Además de ahorrar trabajo, hay que tener presente que el hecho de habernos valido de esa API generada por *Eclipse*, conlleva otra gran ventaja que consiste en que se está trabajando de una forma estandarizada. Esto se traduce en un mayor grado de mantenibilidad, puesto que el código será más fácil de entender para nosotros mismos en un futuro, pero no sólo eso, sino que también lo será para todos aquellos desarrolladores que estén familiarizados con esta opción de generación de código de la que dispone *Eclipse*.

14.4.2 Extracción de la información de un fichero MIF

Tras haber generado todas las clases necesarias mediante *Eclipse*, lo que se necesita hacer es recorrer el archivo MIF para extraer toda la información que resulta relevante en nuestro proyecto.

Existen varias implementaciones de *parsers* para archivos XML escritas en Java, por lo que implementar nosotros mismos esta parte carecería de sentido. Recordemos que un *parser* XML resulta válido para tratar archivos MIF, puesto que poseen la misma estructura en forma de árbol.

Existen diversas implementaciones al margen de la API estándar de Java llamada *Java API for XML*, no obstante, se ha optado por utilizar esa API estándar, básicamente por dos motivos: no introducir dependencias externas en nuestro código y evitar incompatibilidades entre las versiones de Java y las de librerías externas.

El uso de las funciones de la API estándar de Java dedicadas a tratar con archivos XML, nos facilita el trabajo de forma considerable, puesto que crea la estructura arbórea de forma automática y en cada uno de los nodos, nos permite utilizar funciones para acceder a todos sus atributos y nodos hijos.

No obstante, cabe destacar que si bien es cierto que mediante el uso de dicha API se logra reducir el trabajo de forma ostensible, también lo es, que no se convierte en una tarea trivial, en gran parte debido al gran número de relaciones que existen entre los múltiples elementos de un esquema conceptual determinado.

En el ejemplo 14.4, se puede observar cómo se utilizan las funciones proporcionadas por la API de Java y las clases generadas a partir del metamodelo de HL7. Concretamente, se muestra cómo extraer el valor de los atributos *name*, *defaultValue*, *maximumMultiplicity* y *minimumMultiplicity* de un objeto de tipo *Property*, que representa un atributo de una clase determinada. Hay que tener en cuenta que el atributo *node* que aparece, es un objeto Java de tipo *Node* y que en este caso, contiene la información relativa a un nodo de tipo *attribute* de la estructura de los ficheros MIF.

En la primera instrucción del ejemplo, se puede apreciar el uso de la API generada automáticamente por *Eclipse* a partir del metamodelo de HL7. Esta API, implementa el patrón de diseño *Factory Method*, que básicamente, lo que nos permite es crear un objeto de tipo *Factory* que contiene todos los métodos dedicados a la construcción de los elementos que aparecen en el metamodelo. De este modo, con la llamada *factory.createProperty()*, se está llamando al método que crea un objeto del tipo *Property*.

En la segunda instrucción, se pueden observar funciones de la API de Java para tratar con archivos XML, concretamente se está obteniendo el valor de un atributo dado su nombre.

En la tercera, se puede ver el uso de un método *setter* que también ha sido generado a partir del metamodelo.

Una vez explicadas estas instrucciones, las restantes no resultan difíciles de entender, simplemente hay que tener en cuenta que en los casos en que un atributo no es obligatorio, antes de asignar un valor mediante una función *setter* se debe comprobar que no sea nulo.

```
1. Property hl7p = factory.createProperty();
2. String pn =
    node.getAttributes().getNamedItem("name").getNodeValue();
3. hl7p.setName(pn);
4. String defaultv =
    node.getAttributes().getNamedItem("defaultValue").getNodeValue();
5. if (defaultv != null) hl7p.setDefault(defaultv);
6. String minMult =
    node.getAttributes().getNamedItem("minimumMultiplicity").
    getNodeValue());
7. hl7p.setLower(Integer.parseInt(minMult));
8. String maxMult =
    node.getAttributes().getNamedItem("maximumMultiplicity").
    getNodeValue()
9. if (maxMult.compareTo("*") == 0) hl7p.setUpper(-1);
10.else hl7p.setUpper(Integer.parseInt(maxMult));
```

Ejemplo 14.4 Uso de la API de Java para extraer el valor de los atributos de una Property

En el ejemplo 14.5, se puede observar el uso de las funciones que proporciona la API de Java para navegar por los hijos de un determinado nodo. En este caso concreto, se desea asignar a un objeto *Property* su tipo, que tal y como se mostró anteriormente se encuentra en un nodo hijo. Este ejemplo, es una continuación del código anterior, por lo que *node* sigue siendo un objeto de tipo *Node* que representa un nodo de tipo *attribute* y *hl7p* es un objeto de tipo *Property*. Por otro lado, el código relativo a la creación de los tipos de datos es muy extenso, por lo que no se muestra al completo, sólo el caso del tipo *INT*.

```
NodeList nl = node.getChildNodes();
for (int i = 0; i < nl.getLength(); ++i) {
  if (nl.item(i).getNodeName().compareTo("type") == 0) {
    String typeName =
     nl.item(i).getAttributes().getNamedItem("name").getNodeValue();
    if (typename.compareTo("INT") == 0) {
      INT dt = factory.createINT();
      dt.setName("INT");
    }
    else if (…)
    else if (…)
    hl7p.setType(dt);
  }
}
```

Ejemplo 14.5 Funciones de la API de Java para navegar por los nodos

La estructura de los archivos MIF, presenta dos características que complican bastante el código del *parser* resultante.

La primera de ellas, es que se pueden encontrar referencias a elementos que aún no se han creado, lo que impide que con un solo recorrido del árbol se pueda extraer toda la información. Imaginemos que se están tratando los extremos de una asociación, si uno de ellos es una referencia a un elemento que aún se desconoce, no podremos asignarle un tipo a la *property* de ese extremo de la asociación. Por este motivo, deben crearse las clases, *choices* y CMETs que aparecen en un fichero MIF, antes de proceder a recorrer su estructura arbórea.

El segundo, consiste en que los tipos de las clases, si sólo se trata el fichero MIF que representa el esquema que se está transformando, sólo se pueden conocer consultando la información del esquema gráfico, ya que la forma de la figura representada nos lo indica. Es una forma poco elegante de resolver el problema, pero es que además no sirve para CMETs y *choices*.

El tipo de los *choices*, podría deducirse a partir de los elementos que contiene, pero al igual que en el caso anterior, podemos encontrar referencias a ellos en lugar de los propios elementos, lo que nos obliga, como antes, a recopilar toda la información referente a los tipos de las clases, antes de recorrer la estructura arbórea.

En cuanto al tipo de los CMETs, no hay forma de deducirlo únicamente a partir del MIF que hace referencia al esquema que se está tratando. Es necesario consultar otro fichero MIF del estándar llamado *cmetinfo.mif*, en el que se encuentra la información de todos los CMETs que se utilizan en los esquemas conceptuales definidos en el estándar.

Con lo que respecta al tipo de las clases, en lugar de optar por la solución anterior, se puede consultar el fichero *MIFStructuredVocabulary.mif*, también proporcionado en el estándar, y que indica el tipo de clase que corresponde a cada posible valor de los atributos *classCode* y *typeCode*, que están presentes en todas las clases.

Este par de inconvenientes, provocan que sea necesario tratar con varios ficheros MIF y que haya que realizar varios recorridos por su estructura. Esto añade cierta complejidad al *parser* y por ello, se ha tomado la decisión de dividir su código en cuatro clases:

- MIFParser.java
- CMETInfoParser.java
- ElementTypesParser.java
- VocabularyParser.java

A continuación, se procede a explicar los detalles más relevantes de cada una de ellas.

14.4.2.1 MIFParser.java

Esta es la clase principal donde se encuentra el grueso del código del *parser*. En este apartado no se ahonda en sus detalles, puesto que básicamente se encarga de recorrer la estructura de los ficheros MIF, que ya fue explicada anteriormente.

Tan sólo resta por explicar que está programada de tal manera, que puede recibir por parámetro la ruta de varios ficheros MIF. Esto resulta muy cómodo, ya que el número de ficheros MIF de la versión 2009 del estándar se sitúa en torno a los 380 e ir ejecutando la aplicación por cada uno de ellos sería una tarea muy pesada.

14.4.2.2 VocabularyParser.java

Esta es la clase Java que lidia con los posibles valores de los atributos *typeCode* y *classCode*. Cada una de las clases de HL7 contiene uno de esos dos atributos de forma obligatoria, y a partir de sus valores, es posible deducir el tipo (*Act*, *Role*, *Entity*, etc) de la clase que los contiene.

Básicamente, esta clase Java consta de siete vectores, uno por cada uno de los tipos de clases definidos en el metamodelo de HL7, e implementa un recorrido por la estructura arbórea del fichero llamado *MIFStructuredVocabulary.mif*, con el objetivo de recopilar en esos vectores los posibles valores que pueden tomar los atributos *typeCode* y *classCode*.

El método de esta clase que recopila la información anteriormente descrita, es llamado una única vez desde la clase principal *MIFParser.java*, antes de empezar a recorrer las estructuras de los ficheros MIF que se le hayan pasado por parámetro.

En el código del ejemplo 14.6, se encuentra un pequeño fragmento del fichero *MIFStructuredVocabulary.mif*, pero que resulta suficiente para explicar su estructura. En este fichero, se encuentran dos tipos de nodos que resultan relevantes en nuestro caso:

- mif:vocabularyTableContents, que indica a qué tipo de clase pertenecen los códigos contenidos en sus nodos hijos y que a la vez es un posible valor de los atributos *typeCode* y *classCode*. En el ejemplo mostrado, el tipo indicado por este nodo es *Participation*.
- mif:vocabularyDomain, cuyo atributo *name* constituye un posible valor de los atributos *typeCode* y *classCode*.

```
<mif:vocabularyTableContents name="ParticipationType">
  <mif:property
   name="Name:Act:participation:Participation">
   participant</mif:property>
  <mif:property
   name="Name:Role:participation:Participation">
   participation</mif:property>
  <mif:vocabularyDomain name="ParticipationParticipation"
   mnemonic="PART">
    <mif:vocabularyDomain name="ParticipationAncillary">
      <mif:property
       name="Name:Act:participation:Participation">
       ancillaryRole</mif:property>
      <mif:property
       name="Sort:Act:participation:Participation">
       F________</mif:property>
      <mif:property
       name="Name:Role:participation:Participation">
       participation</mif:property>
      <mif:property
       name="Sort:Role:participation:Participation">
       F________</mif:property>
      <mif:vocabularyCode mnemonic="ADM">
        <mif:property
         name="Name:Act:participation:Participation">
         admitter</mif:property>
        <mif:property
         name="Sort:Act:participation:Participation">
         FA_______</mif:property>
        <mif:property
         name="Name:Role:participation:Participation">
         admission</mif:property>
        <mif:property
         name="Sort:Role:participation:Participation">
         FA_______</mif:property>
      </mif:vocabularyCode>
      <mif:vocabularyCode mnemonic="ATND">
        <mif:property
         name="Name:Act:participation:Participation">
         attender</mif:property>
        <mif:property
         name="Sort:Act:participation:Participation">
         FB_______</mif:property>
       <mif:property
        name="Name:Role:participation:Participation">
        attenderOf</mif:property>
       <mif:property
        name="Sort:Role:participation:Participation">
        FB_______</mif:property>
      </mif:vocabularyCode>
         …
```

Ejemplo 14.6 Fragmento del fichero MIFStructuredVocabulary.mif

Tal y como se puede deducir tras haber analizado la estructura que presenta el fichero *MIFStructuredVocabulary.mif*, la implementación de la clase *Vocabulary.java* resulta bastante simple, pues lo único que tiene que hacer es recorrer la estructura arbórea mostrada y cuando

se tope con uno de los dos tipos de nodos señalados anteriormente, guardar el valor que toma su atributo *name*.

14.4.2.3 ElementTypesParser.java

Tal y como se afirmó anteriormente, es necesario crear todos los *choices* y clases antes de analizar la estructura de un determinado fichero MIF. La clase *ElementTypesParser.java*, se encarga precisamente de eso, recorre toda la estructura de un MIF tratando sólo los nodos relativos a clases y *choices*. Al final de ese recorrido, habrá creado dos listas que contendrán todas las clases y *choices* presentes en el esquema respectivamente.

Esta clase, utiliza las funciones definidas en la anterior, *VocabularyParser.java*, ya que al crear una clase necesita saber su tipo y no lo puede obtener directamente, sino que a partir de los valores de los atributos *classCode* y *typeCode* que va encontrando, consulta a *VocabularyParser.java* cuál es el tipo que les corresponde.

14.4.2.4 CMETInfoParser.java

Del mismo modo que necesitamos saber qué clases y *choices* existen en el esquema antes de empezar a analizar su estructura, también debemos conocer que CMETs contiene por el mismo motivo. La clase *CMETInfoParser.java*, se encarga de crear de antemano todos los CMETs presentes en el fichero MIF que se está analizando.

La información que contiene el metamodelo de HL7 diseñado sobre los CMETs, no puede extraerse directamente del fichero MIF que representa un diagrama concreto. Para obtenerla, resulta necesario consultar el fichero MIF del estándar llamado *cmetinfo.coremif*, que contiene la información de todos los CMETs que se utilizan en los esquemas conceptuales del estándar.

En el ejemplo 14.7, se puede observar cómo se define la información de los CMETs en ese fichero. Concretamente, se muestra el caso del CMET con nombre *E_PersonUniversal*, mostrado en la figura 14.15.

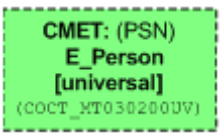


Figura 14.15 CMET E_Person, obtenido de [Hl7Ball]

```
<mif:ownedCommonModelElement name="E_PersonUniversal"
 attributionLevel="universal" entryKind="Entity">
  <mif:supplierStructuralDomain mnemonic="PSN"/>
   <mif:annotations>
      <mif:description>
         <mif:text>Used to completely specify information about a
          person"</mif:text>
      </mif:description>
   </mif:annotations>
   <mif:specializationChildStaticModel root="DEFN" section="IM"
    subSection="CO" domain="CT" artifact="MT-deprecated"
    realm="UV" version="01" id="030200"/>
   <mif:specializationChildEntryClass name="Person">
      <mif:supplierStructuralDomain mnemonic="PSN"/>
   </mif:specializationChildEntryClass>
</mif:ownedCommonModelElement>
```

Ejemplo 14.7 Definición del CMET E_Person en el fichero cmetinfo.coremif

Recordemos que según lo descrito en el metamodelo de HL7, se necesitan conocer los siguientes atributos de un CMET: *name*, *rootClassCode*, *attributionLevel*, *identifier* y *mainElementType*. Analizando el fragmento anterior, es fácil comprobar que de él se pueden extraer todos ellos:

- *name*, corresponde al atributo *name* que se encuentra dentro del nodo *mif:ownedCommonModelElement*.
- *rootClassCode*, se encuentra en el atributo *mnemonic* del nodo *mif:supplierStructuralDomain*.
- *attributionlevel*, se encuentra dentro del nodo *mif:ownedCommonModelElement*.
- *identifier* es el más complicado, puesto que es el resultado de concatenar varios atributos pertenecientes al nodo *mif:specializationChildStaticModel*. Concretamente, es el resultado de *subSection* + *domain* + "_MT" + *id* + *realm* + *version*, donde el símbolo "+" representa la concatenación entre dos *strings*. Así, el identificador del CMET mostrado en el ejemplo es *COCT_MT030200UV01*.
- *mainElementType*, corresponde al valor del atributo *entryKind* del nodo *mif:ownedCommonModelElement*.

14.4.3 Creación del fichero XML

Una vez se han extraído todos los datos del fichero MIF que se está analizando en ese momento, éstos deben ser exportarlos a un fichero XML. De nuevo, en este aspecto no debemos reinventar la rueda, ya existen librerías que permiten exportar objetos java a ficheros

XML y por tanto las aprovecharemos en nuestro proyecto. Para ello, se ha seguido un tutorial creado por Lars Vogel que se puede encontrar en [Vogel].

Las instrucciones que nos permiten realizar esa tarea se pueden encontrar en el código del ejemplo 14.8.

```
Resource.Factory.Registry reg = Resource.Factory.Registry.INSTANCE;
Map<String, Object> m = reg.getExtensionToFactoryMap();
m.put("hl7", new XMIResourceFactoryImpl());
ResourceSet resSet = new ResourceSetImpl();
Resource resource = resSet.createResource(URI.createURI("hl7/" +
fileName + ".hl7"));
//---resto del código del parser
resource.save(Collections.EMPTY_MAP);
```

Ejemplo 14.8 Código Java para crear un fichero XML

La idea es que en el objeto de tipo Resource, deben guardarse todos los otros objetos que queremos que aparezcan en el XML. Esto lo hacemos mediante la instrucción *resource.getContents().add(obj),* donde *obj* es el objeto que se desea guardar. Una vez se ha terminado de recorrer la estructura arbórea del fichero, basta con ejecutar la instrucción *resource.save(Collection.EMPTY_MAP)* para que todos esos objetos queden plasmados en el archivo indicado anteriormente con la llamada al método *createResource()*.

14.5 Resultado de ejemplo

En esta sección, se explican los detalles del fichero XML que se obtiene como resultado de ejecutar el *parser* pasando como parámetro el fichero MIF correspondiente al esquema del estándar con identificador *COCT_MT030000UV08*, incluido dentro del dominio de *Patient Administration* (figura 14.16). La versión gráfica de este esquema, se muestra en la figura 14.17.

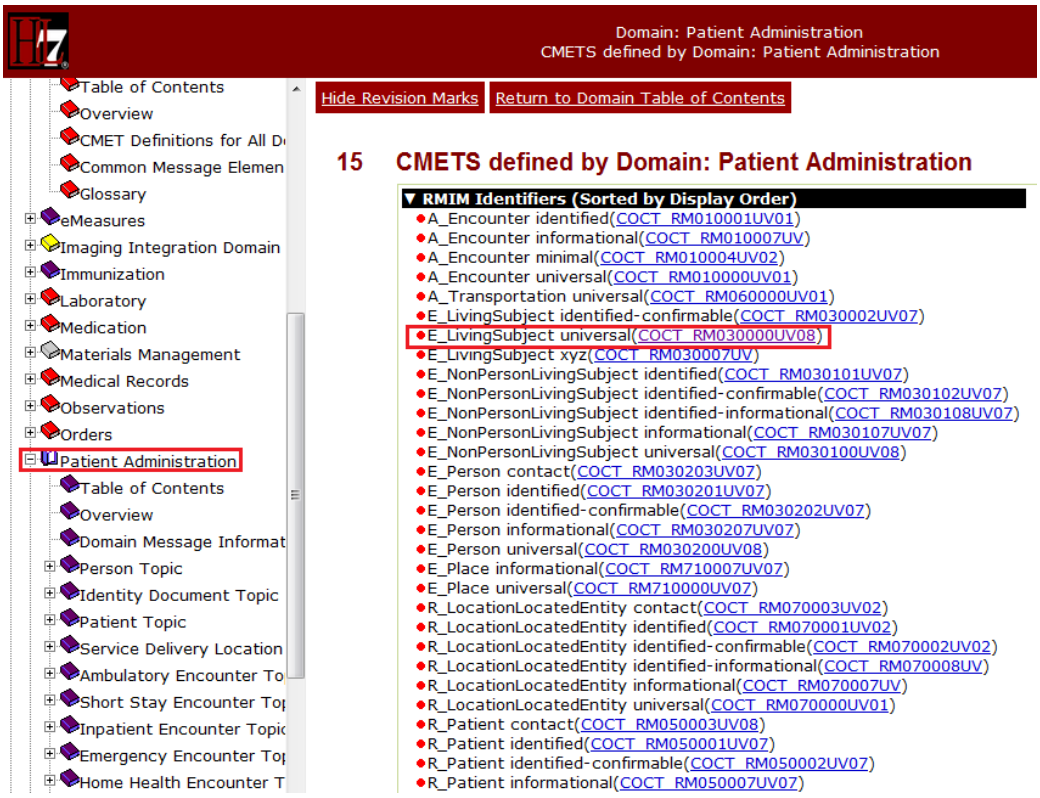


Figura 14.16 Localización del esquema E_LivingSubject universal dentro del ballot

Creemos que es un buen ejemplo, puesto que su tamaño es relativamente reducido y los conceptos que presenta no resultan difíciles de entender. Existen algunos tipos de elementos que aparecen en el metamodelo de HL7, pero que no están presentes en este esquema. Para solventar este problema, en el capítulo 19 se muestra un ejemplo de transformación que los contiene.

En el esquema conceptual ejemplificado, se encuentran todos los conceptos relacionados con los seres vivos. El elemento focal, en este caso, es un *choice* llamado *EntityChoiceSubject*, que contiene dos clases de tipo *Entity*: *Person* y *NonPersonLivingSubject*.

En cuanto a la información sobre personas, vemos que en el esquema se reflejan, entre otros, los idiomas que domina (*LanguageCommunication*), si es estudiante (*Student*) o si trabaja (*Employment*) y en caso de que así sea, en qué organización (*E_Organization*). Toda esta serie de conceptos sólo se relacionan con la clase *Person* puesto que no aplican en otro tipo de seres vivos como pueden ser plantas o animales, representados por la clase (*NonPersonLivingSubject*).

No obstante, a parte de estos, se pueden encontrar toda otra serie de conceptos que sí que aplican a ambos, tales como el lugar de nacimiento (*BirthPlace*) y las personas (*E_Person*) u organizaciones (*E_organization*) que poseen su custodia (*Guardian*).

Figura 14.17 Esquema E_LivingSubject, obtenido de [Hl7Ball]

El código XML resultado de ejecutar la aplicación que se ha construido en esta fase del proyecto, pasando como parámetro el fichero MIF que representa el esquema expuesto anteriormente, no se presenta en esta sección en su totalidad puesto que es bastante extenso.

El objetivo de esta sección, consiste en mostrar pequeños fragmentos de ese fichero XML, pero que sean lo suficientemente representativos como para que a partir de ellos se pueda entender cualquiera de los ficheros XML que la herramienta desarrollada arroja como resultado.

Para empezar, en el ejemplo 14.9 se puede observar el nodo XML que hace referencia a la clase *Person*.

```
<hl7_metamodel:EntityType name="Person" ownedAttribute="/22 /23 /25
/27 /29 /30 /32 /34 /36 /37 /38 /40 /41 /42 /43 /45 /46 /47 /49
/51"/>
```

Ejemplo 14.9 Nodo XML referente a la clase Person

Todas las clases presentan la misma estructura en el fichero XML. En primer lugar, el nombre del nodo es hl7_metamodel:*tipoClase*, en este caso, al tratarse de una clase de tipo *Entity*, es hl7_metamodel:*EntityType*.

En cuanto a los atributos del nodo, existen dos. El primero es *name*, que indica el nombre de la clase en cuestión y el segundo es *ownedAttribute*, que representa el conjunto de atributos que le pertenecen.

En cuanto a estos últimos, podemos ver que no aparece el nodo completo, sino que se referencian mediante un identificador numérico. Cada elemento del fichero XML tiene asignado un identificador de este tipo, que corresponde al orden en el que aparecen declarados en dicho fichero. Al primer elemento le corresponde el número “0”, al segundo el “1” y así sucesivamente.

Hay que tener claro que este identificador no coincide con el número de línea, debido a que los dos primeros elementos, que no son propios del esquema que se está representando sino que aparecen en cualquier fichero XML, no se cuentan.

En el ejemplo 14.10, se pueden ver los primeros elementos del ejemplo que estamos tratando y el identificador que corresponde a cada uno de ellos.

```
1. <?xml version="1.0" encoding="ASCII"?>
2. <xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:hl7_metamodel="http://hl7_metamodel/1.0">
```

```
3.    <hl7_metamodel:EntityType name="Person" ownedAttribute="/22 /23
       /25 /27 /29 /30 /32 /34 /36 /37 /38 /40 /41 /42 /43 /45 /46
       /47 /49 /51"/> (elemento con id = 0)
4.    <hl7_metamodel:RoleType name="Employment" ownedAttribute="/54
       /56 /57 /59 /61 /62 /64 /65 /67 /68"/> (elemento con id = 1)
5.    <hl7_metamodel:RoleType name="Citizen" ownedAttribute="/75 /77
       /78"/> (elemento con id = 2)
6.    <hl7_metamodel:RoleType name="Student" ownedAttribute="/85 /87
       /89 /91 /93 /94"/> (elemento con id = 3)
7.    <hl7_metamodel:InfraestructureType name="LanguageCommunication"
       ownedAttribute="/101 /102 /103 /104"/> (elemento con id = 4)
8.    <hl7_metamodel:EntityType name="NonPersonLivingSubject"
       ownedAttribute="/106 /107 /109 /110 /112 /113 /114 /115 /116
       /117 /118 /119 /120 /121 /122 /123 /124 /125"/> (elemento con
       id = 5)
…
   </xmi:XMI>
```

Ejemplo 14.10 Primeros elementos del fichero XML relativo al esquema E_LivingSubject

Por otra parte, en cuanto a los *choices*, su nombre se indica en el atributo *name*, y los elementos que contienen se representan asignando al atributo *ownedElement* el conjunto de identificadores que les corresponden. En el ejemplo 14.11, se encuentra el nodo del XML que representa al *choice EntityChoiceSubject* del esquema ejemplificado. Tal y como se puede comprobar, en *ownedElement* encontramos los identificadores 0 y 5, que pertenecen a las clases *Person* y *NonLivingSubject*, lo mismo que se indica en la versión gráfica del esquema conceptual mostrado anteriormente.

```
<hl7_metamodel:Choice name="EntityChoiceSubject" ownedElement="/0
/5"/>
```

Ejemplo 14.11 Definición XML del Choice EntityChoiceSubject

Los CMETs, por su parte, son fáciles de interpretar, puesto que sólo contienen los atributos propios de la clase CMET que aparece en el metamodelo de HL7: *name*, *rootClassCode*, *attributionLevel*, *identifier* y *mainClassType*. En el ejemplo 14.12, se puede ver el elemento XML correspondiente al CMET con nombre *E_PlaceUniversal.*

Es importante señalar que no aparecen los elementos que contiene, puesto que el *parser* analiza los ficheros MIF uno por uno y en el que representa al esquema que contiene el CMET, no aparecen los elementos que incluye. Estos elementos, se pueden consultar en el esquema referenciado por el identificador que contiene el CMET en su atributo *identifier*. Es la forma más lógica de hacerlo, puesto que si se decidiese expandir todos los CMETs incluidos en un esquema concreto, el tamaño del mismo se incrementaría considerablemente haciendo muy difícil su lectura.

```
<hl7_metamodel:CMET name="E_PlaceUniversal" rootClassCode="PLC"
attributionLevel="universal" identifier="COCT_MT710000UV01"
mainClassType="Entity"/>
```

Ejemplo 14.12 Definición XML del CMET E_PlaceUniversal

Respecto a los *Entry Points*, encontramos sus atributos *name*, *identifier* y *description*, además de una referencia a la clase foco del esquema, indicada mediante el valor que toma el atributo *choosableElement* del nodo *EntryPoint*, que vuelve a ser un identificador. En el ejemplo 14.13, se encuentra el elemento XML correspondiente al *Entry Point* del esquema que estamos tratando.

```
<hl7_metamodel:EntryPoint name="E_LivingSubjectUniversal"
identifier="COCT_MT030000UV04" description="26 Nov 2003 CMET used to
specify information for a particular person or non-person living
subject such as veterinary subject, plant, etc. "
choosableElement="/13"/>
```

Ejemplo 14.13 Definición XML del Entry Point del esquema E_LivingSubject

En cuanto a los tipos de datos, sólo tienen un atributo *name*. En el ejemplo 14.14, se puede ver cómo se expresan en XML los tipos de datos entero (*INT*) y *Postal Adress* (*AD*).

```
<hl7_metamodel:INT name="INT"/>
<hl7_metamodel:AD name="AD"/>
```

Ejemplo 14.14 Definición XML de los tipos de datos INT y AD

Las asociaciones, como ya se ha visto anteriormente, son un poco más complejas de representar, puesto que además de la asociación en sí, deben definirse las dos *properties* de sus extremos y asignarles un tipo. En el ejemplo 14.15, se puede apreciar cómo se define en XML la asociación existente en el diagrama entre las clases *Person* y *Student*. Para que el ejemplo resulte más fácil de entender, se indica el identificador de cada uno de los elementos involucrados.

En el ejemplo mostrado, se puede comprobar que en el nodo correspondiente a la asociación, sólo se definen los identificadores de sus dos extremos, que en este caso son el 79 y el 89. Por otro lado, en cada una de esas dos *properties* se define el nombre de rol (*name*), el identificador que referencia su tipo (*type*) y su multiplicidad (*upper* y *lower*). En cuanto a esta última, cabe destacar que si el límite inferior (*lower*) es 0 no aparece en el XML, tal y como se puede comprobar en el ejemplo.

```
<hl7_metamodel:EntityType name="Person" ownedAttribute="/22 /23 /25
 /27 /29 /30 /32 /34 /36 /37 /38 /40 /41 /42 /43 /44 /45 /46 /47
 /48"/> (elemento con id = 0)
<hl7_metamodel:RoleType name="Student" ownedAttribute="/80 /81 /82
 /83 /84 /85"/> (elemento con id = 3)
<hl7_metamodel:Association memberEnd="/89 /79"/>  (elemento con
 id = 78)
<hl7_metamodel:Property name="asStudent" type="/3" upper="-1"/>
 (elemento con id = 79)
<hl7_metamodel:Property name="player" type="/0" upper="1"/>
 (elemento con id = 89)
```

Ejemplo 14.15 Definición XML de la asociación existente entre Person y Student

Para terminar, las *properties* que actúan como atributos de una clase en lugar de jugar el rol de extremos de una asociación, se definen de forma muy similar a las de este último caso. Sólo existen dos diferencias, la primera es que el elemento referenciado en el atributo *type* es un tipo de dato en lugar de una clase, *choice* o CMET, y la segunda, que los atributos pueden constar de un valor por defecto indicado por *defaultValue*, y aquellos que son de tipo código, además pueden constar de los atributos *codingStrength*, *mnemonic*, *codeSystemName* y *domainName.* En el ejemplo 14.16, se muestra la definición de todos los atributos de la clase *Student*, además de sus tipos de datos.

```
<hl7_metamodel:RoleType name="Student" ownedAttribute="/80 /81 /82
 /83 /84 /85"/> (elemento con id = 3)
<hl7_metamodel:CS name="CS"/> (elemento con id = 21)
<hl7_metamodel:II name="II"/> (elemento con id = 24)
<hl7_metamodel:AD name="AD"/> (elemento con id = 54)
<hl7_metamodel:TEL name="TEL"/> (elemento con id = 56)
<hl7_metamodel:IVL_TS name="IVL_TS"/> (elemento con id = 59)
<hl7_metamodel:Property name="classCode" type="/21" upper="1"
 lower="1" codingStrength="CNE" mnemonic="STD"
 codeSystemName="RoleClass"/>  (elemento con id = 80)
<hl7_metamodel:Property name="id" type="/24" upper="-1"/>
 (elemento con id = 81)
<hl7_metamodel:Property name="addr" type="/54" upper="-1"/>
 (elemento con id = 82)
<hl7_metamodel:Property name="telecom" type="/56" upper="-1"/>
 (elemento con id = 83)
<hl7_metamodel:Property name="statusCode" type="/21" upper="-1"
 codingStrength="CNE" domainName="RoleStatus"/>  (elemento con
 id = 84)
<hl7_metamodel:Property name="effectiveTime" type="/59"
 upper="1"/>  (elemento con id = 85)
```

Ejemplo 14.16 Definición XML de la clase Student y sus atributos

14.6 Resumen

A lo largo de este capítulo, se ha justificado la elección de los ficheros MIF como archivos fuente para la extracción de información de los modelos y se han expuesto los detalles de su estructura. Además, se han explicado todos los detalles relativos al desarrollo de la herramienta que permite, a partir de esos ficheros MIF, generar instancias escritas en XML del metamodelo de HL7 que se diseñó en la fase anterior del proyecto. Finalmente, también se ha mostrado la estructura que poseen esos archivos XML resultantes y se han explicado sus características más relevantes.

Llegados a este punto, tenemos el metamodelo de HL7 diseñado, así como las instancias XML de ese metamodelo que representan el conocimiento de todos los esquemas conceptuales del estándar que deseamos transformar. Esto significa que ya se poseen todos los elementos necesarios para poder empezar a definir las reglas de transformación de ATL, que es precisamente lo que se explica a lo largo de los dos capítulos siguientes.

15 ATL

Después de haber diseñado el metamodelo de HL7 y haber obtenido las instancias XML de ese metamodelo, correspondientes a todos los esquemas del estándar que se desean convertir, se cumplen todos los requisitos para poder empezar a trabajar con la herramienta de transformación de modelos ATL.

En este capítulo, se justifica la elección de ATL, se hace una breve introducción al lenguaje explicando sus características principales y se presenta un ejemplo de conversión sencillo.

15.1 Justificación de la elección de ATL

Lo cierto es que no hay demasiados lenguajes para trabajar en el ámbito de las transformaciones M2M (*model-to-model*).

A día de hoy se podría decir que el más utilizado y mejor posicionado es, sin lugar a dudas, ATL. El hecho que mejor lo demuestra es que es la solución estándar incorporada por el proyecto *Eclipse M2M* que es el que contiene todos los *plugins* necesarios para trabajar en temas de modelización conceptual sobre esa plataforma. La página web de dicho proyecto puede encontrarse en [EclM2M].

ATL, tal y como se explica a lo largo de este capítulo, contiene múltiples características que lo convierten en la mejor opción en nuestro caso: realiza las conversiones a partir de metamodelos, por lo que podremos aprovechar el de HL7 diseñado, utiliza una sintaxis próxima a OCL que permite construir consultas complejas, es un proyecto activo, y se puede afirmar que posee una documentación muy elaborada.

A pesar de lo expuesto en los dos párrafos anteriores, antes de empezar a trabajar con ATL se decidió buscar alternativas a dicho lenguaje. En [Biehl], se encuentran descritas las principales herramientas de transformaciones M2M. Entre ellas, se pueden destacar algunas como las siguientes:

- **QVT**. No es un software, sino una especificación en la que se basan herramientas como ATL.
- **QVT Smart**. Es una implementación de QVT pero que lleva sin actualizarse desde 2007 y sin apenas documentación.
- **ModelMorf.** Es una herramienta propietaria y con escasa documentación.

- **Henshin**. Las reglas que permite definir son muy simples y no son suficientes para lo que deseamos hacer en este proyecto.

Como se puede comprobar, todas ellas presentan inconvenientes que, en nuestro caso, convierten a ATL en la mejor opción disponible.

15.2 Características del lenguaje

ATL es una herramienta de transformación de modelos construida sobre *Eclipse*, que permite, dados dos metamodelos MM1 y MM2, una instancia de MM1 escrita en XML y una serie de reglas de conversión de MM1 hacia MM2, generar una instancia de MM2 también escrita en XML, la cual contiene la información transformada de la instancia de MM1 una vez se aplican las reglas para pasar de MM1 a MM2.

En la figura 15.1, se muestra un esquema aclaratorio. En él, los elementos azules son los que debemos proporcionar a la herramienta, y el rojo el que se produce como salida.

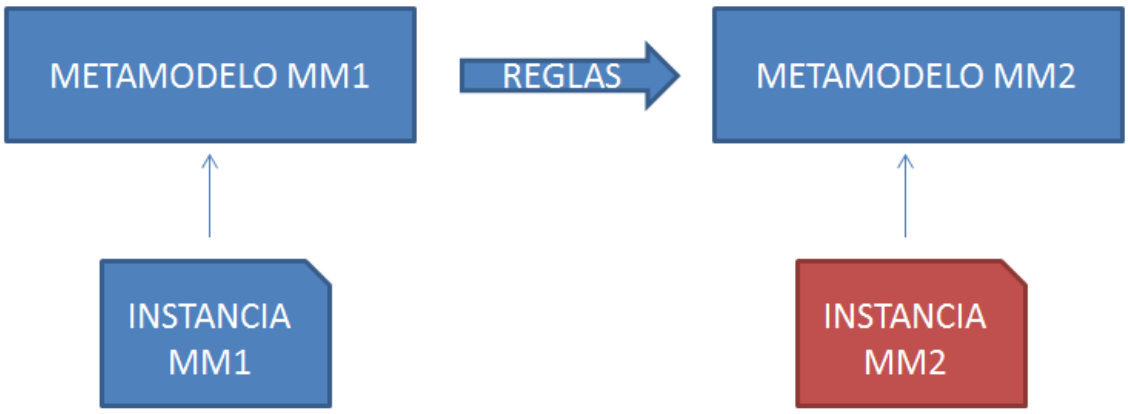


Figura 15.1 Esquema del funcionamiento de ATL

Al lenguaje soportado por la herramienta ATL para la definición de reglas de transformación, también se le conoce como ATL. Este lenguaje se considera híbrido, pues permite el uso tanto de construcciones imperativas como declarativas. Recordemos que en un lenguaje de tipo imperativo, se necesita que el programador controle el flujo del programa, es decir, que provea un algoritmo de forma explícita. Por otro lado, en los de tipo declarativo, el programador define qué es lo que el programa debe hacer declarando condiciones, restricciones, etc., pero no indica exactamente cómo.

A cada uno de los ficheros que contienen las reglas de transformación ATL para pasar de un metamodelo a otro, se los conoce con el nombre de módulos. En ellos, se pueden distinguir los siguientes elementos:

- Una cabecera, en la que se definen los metamodelos y modelos que intervienen en la transformación.
- Una sección opcional de importación de librerías, en caso que resulten necesarias.
- Un conjunto de métodos, similares a los que se tienen en lenguajes como C++ o Java, que en ATL se conocen con el nombre de *helpers*.
- El conjunto de reglas de transformación que permiten convertir instancias de uno de los metamodelos a instancias del otro.

En el ejemplo 15.1, se muestra una posible manera de cómo se declararía la cabecera de un módulo ATL si por ejemplo, se tratase el caso de convertir los tipos de datos de UML en tipos de datos de Java.

```
module UMLDT2JavaDT;
create OUT : JavaDT from IN : UMLDT;
```

Ejemplo 15.1 Cabecera de un módulo de ATL

El nombre del módulo aparece detrás de la palabra clave *module*, en el caso mostrado es *UMLDT2JavaDT*, mientras que los modelos y metamodelos utilizados en la transformación, deben indicarse siguiendo la estructura que se puede ver en la segunda línea del ejemplo, es decir, *create OUT : MD from IN : MO*, donde *MD* es el nombre del modelo destino (en el ejemplo *JavaDT*) y *MO* el nombre del modelo de origen. Además, para que esta línea cobre sentido, es necesario haber declarado previamente cuál es el metamodelo origen (*IN*) y cuál el destino (*OUT*). Esto se indica en las configuraciones de ejecución de *Eclipse*, como se detallará más adelante en la guía de usuario del capítulo 20.

En cuanto a los *imports*, se declaran mediante la instrucción *uses librería*. De este modo, si se quisiera importar la librería que contiene las funciones para tratar *strings*, se haría tal y como se indica en el ejemplo 15.2.

```
uses strings;
```

Ejemplo 15.2 Importación de librerías en ATL

Con lo que respecta a los *helpers*, tal y como se ha afirmado anteriormente, su declaración es similar a la de una función Java, por ejemplo. Así, los *helpers* constan de:

- Un nombre.
- El tipo del objeto al que aplica.
- El tipo del objeto que retorna.
- Un conjunto de parámetros, que puede ser vacío.
- Una expresión ATL que define cómo se calcula el resultado a retornar.

Los *helpers* tienen una utilidad similar a la que presentan las funciones de los lenguajes de programación, es decir, hacer el código más legible y evitar que haya fragmentos redundantes.

En el ejemplo 15.3, se muestra la declaración de un *helper* que suma dos números de tipo entero. Tras la palabra clave *context*, aparece el tipo al que aplica la función, después de *def* se puede encontrar el nombre del *helper* y entre paréntesis la información relativa a los parámetros. Después de la lista de parámetros, se encuentra el tipo del objeto que retorna la función, y finalmente, tras el signo de igualdad encontramos la expresión ATL que define cómo se calcula el resultado.

```
helper context Integer def : suma(x : Integer) : Integer = self + x;
```

Ejemplo 15.3 Declaración de un helper ATL

En cuanto a las reglas de transformación, tal y se ha explicado anteriormente existen principalmente dos tipos: las llamadas *matched rules* y las *called rules*.

Las *matched rules* corresponden al tipo de programación declarativa. En el ejemplo 15.4, se muestra una regla de este tipo.

```
rule EstudianteAPersona {
  from
    est : MMEstudiante!Estudiante
  to
    per : MMPersona!Persona (
      nombre <- est.nombre,
      apellidos <- est.apellidos
    )
}
```

Ejemplo 15.4 Declaración de una matched rule en ATL

Para empezar, hay que tener en cuenta que para referirse a una clase determinada se debe usar la siguiente estructura: *NombreDelMetamodelo!NombreDeLaCLase*. Así, cuando en la regla encontramos *MMPersona!Persona*, significa que se está refiriendo a la clase *Persona* que hay declarada dentro del metamodelo que tiene por nombre *MMPersona*.

Como se puede apreciar en el ejemplo mostrado, el nombre de la regla se indica después de la palabra clave *rule*, en este caso es *EstudianteAPersona*. Las *matched rules*, deben contener dos secciones: *from* y *to*. En la primera se define para qué tipos de elementos del metamodelo fuente hay que generar algún elemento destino, y en la segunda se definen el tipo y las propiedades de esos elementos destino.

De este modo, lo que expresa la regla *EstudianteAPersona* del ejemplo que realiza transformaciones de instancias del metamodelo *MMEstudiante* a instancias de *MMPersona*, es que por cada clase de tipo *Estudiante* que se encuentre en la instancia del metamodelo *MMEstudiante*, se debe generar una instancia de *Persona* del metamodelo *MMPersona*, cuyos atributos *nombre* y *apellidos* tomen el mismo valor que sus homónimos de la clase *Estudiante*. Con respecto a esto último, tal y como se puede observar en el ejemplo, cuando se desea asignar un valor a una variable, se utiliza el símbolo '<-'.

Las *called rules*, a diferencia de las *matched rules*, proveen al programador de mecanismos propios de la programación imperativa. Este tipo de reglas se asemejan bastante a los *helpers* definidos anteriormente, ya que para que se ejecuten, es necesario haberlas invocado y además, aceptan parámetros. La diferencia respecto a los *helpers*, es que las *called rules* pueden crear objetos del metamodelo destino para poder ser incluidos en la instancia resultado de la transformación.

En el ejemplo 15.5, se muestra un ejemplo de *called rule*.

```
rule NuevaPersona (_nombre: String, _apellidos: String,
listaPersonasCreadas: Set(String)) {
  to
    p : MMPersona!Persona (
      nombre <- _nombre
    )
  do {
    p.apellidos <- _apellidos;
    listaPersonasCreadas.add(p.nombre + ' ' + p.apellidos);
  }
}
```

Ejemplo 15.5 Definición de una called rule en ATL

En este último ejemplo mostrado, se ha querido crear el mismo tipo de instancia que en el ejemplo anterior de *matched rule*, para así poder apreciar mejor las diferencias entre los dos tipos de reglas ofrecidas por ATL.

En las *called rules*, existen dos secciones diferenciadas: *to* y *do*. En la primera de ellas, se define lo mismo que en su homónima de las *matched rules*, es decir, el tipo del objeto a crear y sus propiedades.

En la sección *do*, en cambio, no se pueden crear objetos, pero sí que es posible asignar valor a cualquiera de los atributos de los objetos creados, llamar a otras *called rules*, etc. En el ejemplo mostrado, se añade el nombre completo de la persona creada a una lista que se recibe por parámetro. Esta última acción no se podría realizar dentro de la sección *to*, puesto que se modifica una variable externa a la clase que se está creando.

De este modo, en el último ejemplo mostrado, el resultado de su ejecución, es que se crea una instancia de la clase *Persona* del metamodelo *MMPersona* y a sus atributos *nombre* y *apellidos*, se les asignan los valores recibidos por parámetro.

En cuanto al tipo de expresiones que ATL ofrece, se podría decir que son bastante numerosas. Se pueden utilizar operaciones típicas booleanas y numéricas, hacer uso de sentencias condicionales e iterativas, etc.

Lo más destacable es que permite utilizar expresiones de conjuntos muy parecidas a las que ofrece OCL. Así, por ejemplo, la expresión mostrada en el ejemplo 15.6, devolvería todas las instancias de la clase *Persona* del metamodelo *MMPersona* que se llamen Claudia.

```
MMPersona!Persona.allInstances()
        ->select(p | p.name = 'Claudia')
```

Ejemplo 15.6 Expresión OCL usada en una regla ATL

15.3 Ejemplo de transformación

Con el objetivo de consolidar todas las características explicadas sobre ATL y poder comprender mejor las reglas que definirán las transformaciones de HL7 a UML, se considera oportuno exponer un ejemplo adaptado de los que se proporcionan en [ATLw], la página web de ATL.

En el ejemplo, lo que se pretende es pasar de un metamodelo que representa familias, a otro que representa personas. De modo que lo expuesto en el ejemplo 15.7, pase a ser lo que se expresa en el 15.8.

```
Familia García
  Padre: Andrés
```

```
Madre: Berta
Hijo: Carlos
Hija: Diana
```

Ejemplo 15.7 Elementos del metamodelo de familias

```
Sr. Andrés García
Sra. Berta García
Sr. Carlos García
Sra. Diana García
```

Ejemplo 15.8 Elementos del metamodelo de personas

Lo primero que se debe hacer es diseñar ambos metamodelos. El de familias se encuentra en la figura 15.2, mientras que el de personas se muestra en la 15.3.

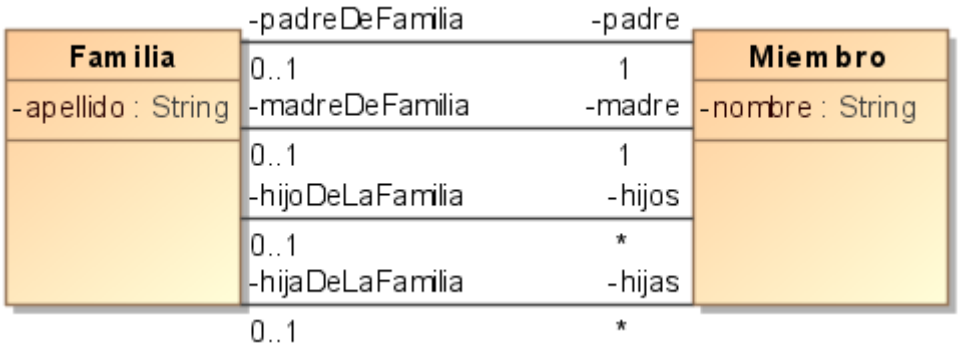


Figura 15.2 Metamodelo de familias

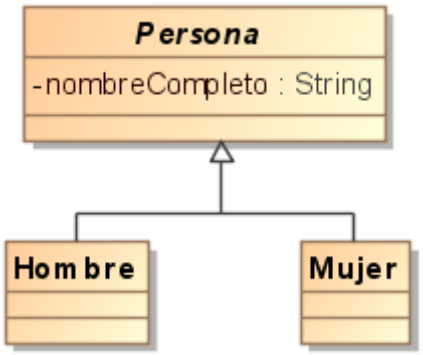


Figura 15.3 Metamodelo de personas

El siguiente paso, consiste en obtener una instancia XML del metamodelo de familias que contenga la información de una familia de la que queramos obtener instancias del metamodelo de personas. En este caso, se utilizará la instancia XML descrita en el ejemplo 15.9, que representa a la familia descrita anteriormente en ejemplo 15.7.

```
<?xml version="1.0" enconding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi=http://www.omg.org/XMI
xmlns="Familias">
  <Familia apellido="García">
    <padre nombre="Andrés">
    <madre nombre="Berta">
    <hijos nombre="Carlos">
    <hijas nombre="Diana">
  </Familia>
</xmi:XMI>
```

Ejemplo 15.9 Instanciación XML del metamodelo de familias

Como se puede comprobar, la estructura del fichero XML mostrado es ligeramente diferente a la de los ficheros XML instancias del metamodelo de HL7 mostrados en el capítulo 14, ya que en el ejemplo de la web de ATL no se usan identificadores numéricos para referenciar objetos, sino que se incluyen otros elementos dentro de una etiqueta más general. Esto no presenta ningún problema ni nos impide utilizar nuestros ficheros XML con reglas ATL. Son simplemente diferentes formas de describir instancias en XML, pero igualmente válidas.

Una vez se tienen las instancias XML que se desean convertir y ambos metamodelos, se está en disposición de poder definir las reglas de transformación en un módulo ATL, que se explica a continuación.

15.3.1 Cabecera

La cabecera del módulo, se define tal y como se muestra en el ejemplo 15.10.

```
Module FamiliasAPersonas;
Create OUT : Personas from IN : Familias;
```

Ejemplo 15.10 Definición de la cabecera ATL en el ejemplo de familias y personas

15.3.2 Helpers

Para facilitar la definición de las reglas de conversión, en este caso, se ha optado por utilizar un par de *helpers*. El primero, retorna un booleano, indicando si un miembro de la familia es mujer o no (ejemplo 15.11), y el segundo, a partir de un miembro, retorna un *string* que representa el apellido de su familia (ejemplo 15.12). En estos *helpers*, aparecen llamadas a la función *oclIsUndefined()*, que retorna lo mismo que en OCL, es decir, un booleano que toma el valor de verdadero cuando el objeto mediante la cual es invocada existe y falso en caso contrario.

```
helper context Familias!Miembro def: esMujer() : Boolean =
  if not self.madreDeFamilia.oclIsUndefined() then true
  else
    if not self.hijaDeLaFamilia.oclIsUndefined() then true
    else false
    endif
  endif;
```

Ejemplo 15.11 Definición del helper esMujer() en el ejemplo de familias y personas

```
helper context Familias!Miembro def: apellidoFamilia() : String =
  if not self.padreDeFamilia.oclIsUndefined() then
    self.padreDefamilia.apellido
  else
    if not self.madreDeFamilia.oclIsUndefined() then
      self.madreDeFamilia.apellido
    else
      if not self.hijoDeLaFamilia.oclIsUndefined() then
        self.hijoDeLaFamilia.apellido
      else
        self.hijaDeLaFamilia.apellido
      endif
    endif
  endif;
```

Ejemplo 15.12 Definición del helper apellidoFamilia() en el ejemplo de familias y personas

15.3.3 Reglas de transformación

En este ejemplo, se definen dos reglas de transformación. La primera es la que se muestra en el ejemplo 15.13. En ella, la condición que encontramos en la sección *from*, incluye a todos los miembros de familias que no sean mujeres, vemos que esto se define en una sola línea gracias a haber definido anteriormente el *helper esMujer()*. Por otro lado, en la sección *to*, se crea una instancia de la clase *Hombre* del metamodelo *Personas* y a su atributo *nombreCompleto*, se le asigna la concatenación del nombre del miembro del que proviene y del apellido de su familia.

```
rule MiembroAHombre {
  from
    m: Familias!Miembro (not m.esMujer())
  to
    h: Personas!Hombre (
      nombreCompleto <- m.nombre + ' ' + m.apellido
    )
}
```

Ejemplo 15.13 Regla de transformación MiembroAHombre del ejemplo de familias y personas

La segunda regla es muy parecida a la anterior, aunque en esta ocasión en la sección *from* se incluyen a todos los miembros de familias que sean mujeres y en la *to*, se crea una instancia de *Mujer* en lugar de *Hombre* (ejemplo 15.14).

```
rule MiembroAMujer {
  from
    mi: Familias!Miembro (mi.esMujer())
  to
    mu: Personas!Mujer (
      nombreCompleto <- mi.nombre + ' ' + mi.apellido
    )
}
```

Ejemplo 15.14 Regla de transformación MiembroAMujer del ejemplo de familias y personas

15.3.4 Resultado

En el ejemplo 15.15, se muestra el fichero XML resultante de la conversión. La información contenida es la que ya se mostró anteriormente.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xmi:XMI xmi:version="2.0" xmlns:xmi=http://www.omg.org/XMI
xmlns="Personas">
  <Hombre nombreCompleto="Andrés García" />
  <Hombre nombreCompleto="Carlos García" />
  <Mujer nombreCompleto="Berta García" />
  <Mujer nombreCompleto="Diana García" />
</xmi:XMI>
```

Ejemplo 15.15 Fichero XML resultante de la conversión en el ejemplo de familias y personas

En resumen, a partir de una instancia XML de un metamodelo inicial, se ha obtenido una instancia XML de un metamodelo destino y para ello, antes se han tenido que definir ambos metamodelos, la instancia XML del metamodelo fuente y las reglas de transformación.

16 CONVERSIÓN DE LOS ELEMENTOS HL7 A UML

En este capítulo, se explica todo aquello que tiene que ver con la fase del proyecto en la que se escriben las reglas de transformación de ATL. Concretamente, se muestra cómo se construyen las reglas de conversión para cada uno de los elementos del metamodelo de HL7, teniendo en cuenta las transformaciones que hacen las herramientas ya existentes, y se explican ejemplos de ficheros UML resultantes.

16.1 Herramientas existentes

Antes de proponer nosotros mismos las reglas de transformación, debemos buscar si hay otros programas que ya las hagan y analizar la corrección de sus propuestas. Básicamente, hay tres proyectos que tratan este tema: MDHT, la wiki de HL7 y *HyperModel*. A continuación, se explican las características principales de cada uno de ellos.

16.1.1 MDHT (Model-Driven Health Tools)

MDHT es un software construido sobre la plataforma *Eclipse*. La web del proyecto se puede encontrar en [MDHTw].

MDHT aún no se puede considerar estable. Además, la única conversión que es capaz de hacer a UML, es la del esquema del *Clinical Document Architecture*, que ya fue explicado en el capítulo 13. De este modo, las propuestas de conversión que se analizarán sobre esta herramienta, estarán basadas en ese esquema.

Otro de sus grandes inconvenientes, es que no se puede considerar que sea usable, ya que la documentación que proporciona es escasa y las transformaciones no se pueden realizar de una forma intuitiva.

16.1.2 Wiki de HL7

En una de las páginas de la wiki de HL7, en concreto la que se puede encontrar en [hl7Wiki], se ha abierto un debate sobre la posibilidad de representar los esquemas del estándar HL7 en UML. En ella, se puede encontrar un esquema extraído del estándar y la propuesta que hacen de cómo quedaría transformado a UML.

Por el momento, esta propuesta no ha sido implementada en ningún software, por tanto, a la hora de valorarla, únicamente nos podemos basar en el esquema UML que se puede encontrar en la página de la wiki citada anteriormente.

16.1.3 HyperModel

HyperModel, al igual que MDHT, es un software construido sobre la plataforma *Eclipse*. Su caso es curioso, porque el ejemplo de transformación que se proporciona en la web del proyecto, que se puede encontrar en [HyperMow], no coincide exactamente con los resultados que se pueden obtener ejecutando el programa.

El proyecto lleva mucho tiempo sin actualizarse, por lo que suponemos que el ejemplo mostrado en la web es lo que se quería implementar, pero que por algún motivo no llegó a hacerse. De todas maneras, se analizarán tanto las propuestas que se hacen en la web del proyecto, como los resultados que se obtienen al ejecutar el software.

16.2 Conversiones

En esta sección, se analiza para cada uno de los elementos que se pueden encontrar en el metamodelo de HL7 diseñado, las diferentes propuestas de conversión a UML que se hacen desde las herramientas anteriormente mencionadas y se justifican todas las elecciones tomadas.

16.2.1 Clases

En MDHT, las clases no se estereotipan. Para distinguir las clases que son de diferentes tipos (*Act*, *Role*, etc), lo que se hacees crear un esquema por separado que únicamente contiene las clases principales del estándar y sus relaciones, es decir, un subconjunto del RIM. Así, en los esquemas resultantes de las conversiones, cada una de las clases que aparecen, hereda de una de una de esas principales.

En cuanto al esquema que contiene las clases principales, cabe destacar que la única información que contiene, son las relaciones entre clases. En un principio, se podría pensar que también debería contener los atributos de esas clases, pero sería un error, puesto que tal

y como se definen los mecanismos de refinamiento vistos en el capítulo 10 de este documento, se puede dar el caso que una clase de tipo *Act* contenga menos atributos que la clase *Act* del RIM. Un ejemplo de esto puede apreciarse en la figura 16.1, en la que la clase *Act* extraída del RIM contiene más atributos que la clase *Account* que se puede encontrar en el esquema *A_Account universal* con identificador *COCT_RM110000UV*.

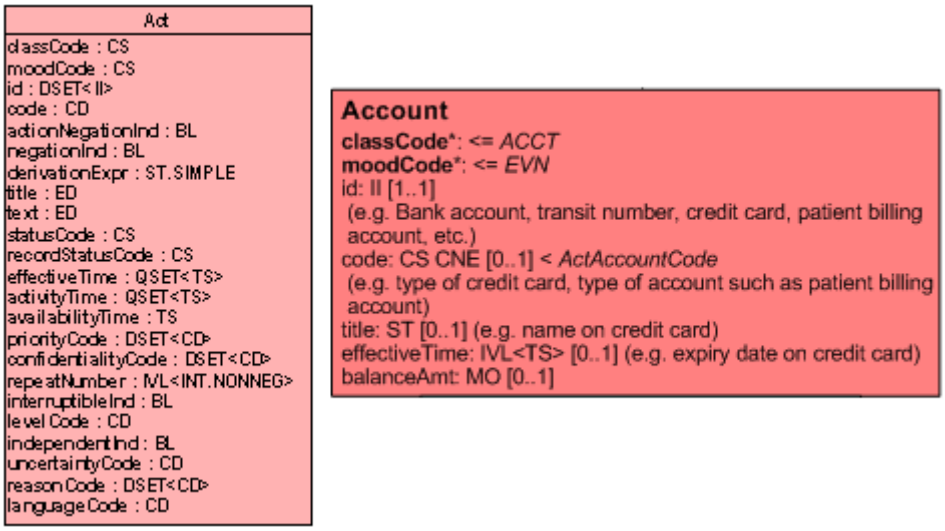


Figura 16.1 Clase Act del RIM y clase Account de tipo Act, obtenidas de [HI7Ball]

En la wiki de HL7, las clases aparecen estereotipadas, algunas según su tipo y otras según los valores del vocabulario que pueden tomar sus atributos *classCode* y *typeCode*, es decir, en la mayoría de casos, los estereotipos que encontramos son *Act*, *Role*, etc., pero por ejemplo, en una de las clases que se muestran en el ejemplo que proporcionan, *Container*, el estereotipo que se le aplica también lleva el nombre de *Container*. Además, cabe destacar que hacen dos propuestas de esquemas UML para un mismo esquema HL7, en una de ellas, las clases de tipo *Participation* aparecen como clases asociativas y en el otro no.

El hecho de representar ese tipo de clases mediante clases asociativas es un error por dos motivos. El primero es que en el RIM no está definido así, y por tanto, se estaría yendo en contra de los mecanismos de refinamiento definidos por el estándar.

El segundo es que podemos encontrar ejemplos de clases de tipo *Participation*, que no pueden ser representadas como clases asociativas según las reglas establecidas por UML.

Para explicar este hecho, se muestra un pequeño fragmento de esquema UML en la figura 16.2. Según lo definido por UML, en ese caso, sólo puede existir una instancia de la asociación

que relaciona las clases A y B por cada pareja de instancias de estas dos clases. Y esto implica que, por cada pareja de instancias de las clases A y B, sólo puede existir una instancia de C.

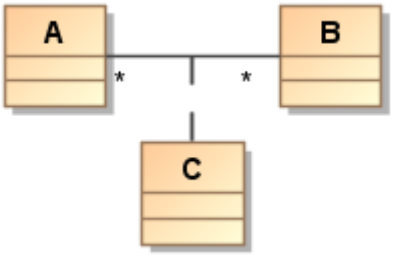


Figura 16.2 Clase asociativa de UML

Esto, llevado al ejemplo de HL7 significa que por cada pareja de instancias de clases de tipo *Act* y *Role*, sólo podría existir una de tipo *Participation.*

No obstante, es posible encontrar ejemplos de esquemas donde ese hecho no se cumple. Así por ejemplo, en el esquema del *Clinical Document Architecture* con identificador *POCD_RM000040* que se muestra en la figura 16.3, se puede ver que una clase de tipo *Act* (*ClinicalDocument*), está relacionada con una de tipo *Role* (*AssignedEntity*) por medio de dos clases distintas de tipo *Participation* (*Authenticator* y *LegalAuthenticator*).

Si la clase *Participation* se considerase asociativa, entonces tanto *Authenticator* como *legalAuthenticator* deberían serlo también, puesto que son subclases de *Participation*. En ese caso, no podrían relacionar los mismos extremos, puesto que por la explicación expuesta anteriormente, dadas dos instancias de los extremos de una asociativa, sólo puede haber una instancia de *Participation*. No obstante, en el ejemplo de la figura 16.3, podría haber dos instancias de *Participation* diferentes, una de *Authenticator* y otra de *LegalAuthenticator*.

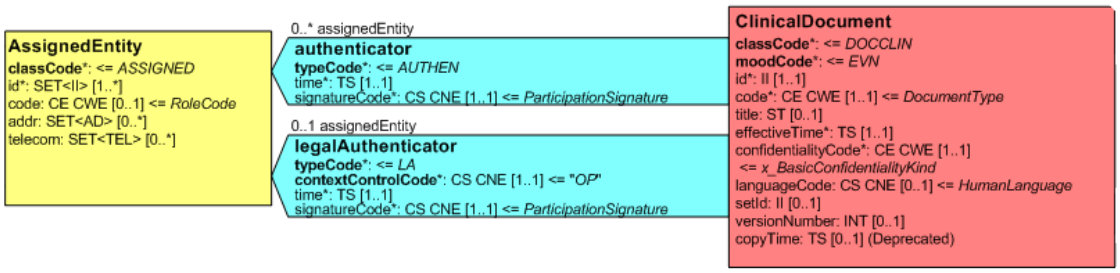


Figura 16.3 Fragmento del esquema Clinical Document Architecture, obtenido de [Hl7Ball]

En cuanto a las transformaciones obtenidas al ejecutar *HyperModel*, las clases no aparecen estereotipadas y es posible distinguir unas de otras porque gracias a la herramienta gráfica de

definición de modelos incorporada en *Eclipse*, de alguna manera se ha definido que cada una de ellas debe aparecer pintada de un color distinto, que corresponde al que se utiliza en el estándar.

Por otro lado, en algunas ocasiones, las clases de tipo *Participation* aparecen como clases asociativas, que es un error por las razones expuestas. En otras, la clase simplemente no aparece, se relacionan directamente las clases de tipo *Act* y *Role*, y *Participation* pasa a ser un simple nombre de rol en esa asociación.

Este último hecho presenta dos inconvenientes, el primero es que al igual que ocurre con el tema de las clases asociativas, no se respeta lo definido en el RIM, y el segundo, es que al eliminar la clase *Participation*, se está perdiendo información, puesto que éstas presentan un conjunto de atributos que no aparecen reflejados en los esquemas UML. En la figura 16.4, se puede ver un fragmento de esquema resultado de aplicar las transformaciones de HyperModel donde se ha usado este mecanismo de supresión sobre el fragmento de esquema original de la figura 16.5, extraído del esquema con nombre *Patient Activate* e identificador *PRPA_RM201301UV*.

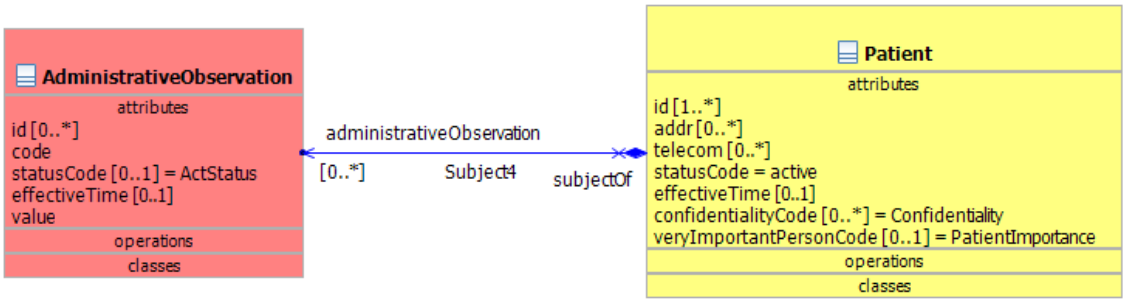


Figura 16.4 Fragmento del esquema Patient Activate donde se suprime una Participation, obtenido de [HyperMw]

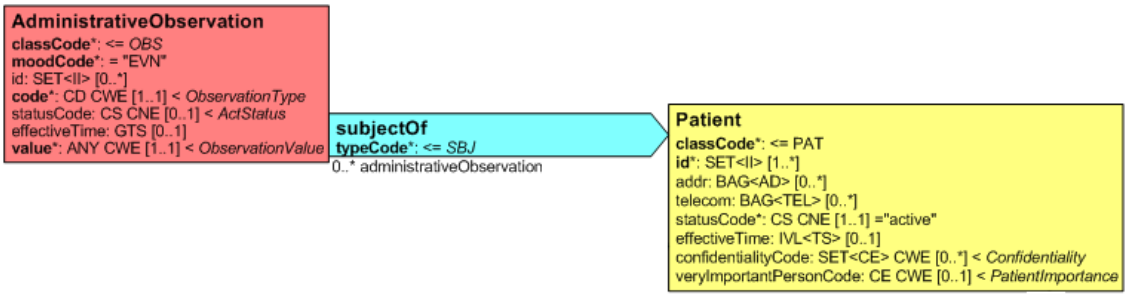


Figura 16.5 Fragmento del esquema Patient Activate, obtenido de [Hl7Ball]

Por otra parte, en el ejemplo mostrado en la web de *HyperModel*, aparecen todas las clases de tipo *Participation* como clases asociativas, y para distinguir los diferentes tipos de clases, se usan estereotipos del mismo modo que en la wiki de HL7.

Por lo que se ha ido justificando, está claro que no podemos expresar las clases de tipo *Participation* como asociativas ni tampoco suprimirlas. La decisión que debemos tomar aquí, consiste en decantarnos por utilizar estereotipos para distinguir los diferentes tipos de clases, o por el contrario, optar por el método empleado por MDHT, es decir, crear un esquema conceptual por separado que contenga las clases del RIM y sus asociaciones y desde el cual todos los demás hereden.

Ambas opciones son correctas, pero hemos optado por el uso de estereotipos. Principalmente, porque queremos evitar introducir dependencias entre esquemas mientras resulte posible. La razón de esto, es que para las personas que vayan a trabajar con los esquemas resultantes, será más cómodo y además, nos facilitará las cosas de cara a las conversiones con ATL. Por otro lado, creemos que el uso de estereotipos puede presentar otras ventajas, por ejemplo, al utilizar herramientas de filtrado de esquemas, puesto que resultaría relativamente sencillo seleccionar todas las clases que sean de un determinado tipo. El nombre de los estereotipos usados corresponden a los tipos de clases existentes en HL7: *Act*, *ActRelationship*, *Participation*, *Role*, *RoleLink*, *Entity* e *Infraestructure*.

16.2.2 Atributos

En todos los proyectos mencionados, los atributos de los esquemas HL7 se traducen directamente a atributos UML.

No obstante, existe un detalle que cabe destacar. Tanto en la propuesta que se hace en la wiki de HL7, como en el ejemplo que se puede consultar en la web de *HyperModel*, no se incluyen los atributos *typeCode* ni *classCode*. El motivo es que estos atributos pueden deducirse a partir del nombre de la clase si se consulta en los *ballots* la jerarquía de conceptos definida y por tanto, son considerados redundantes.

En cuanto a la información relativa a los valores del vocabulario que los atributos pueden tomar, recordemos que había cuatro: *domainName*, *codeSystemName*, *mnemonic* y *defaultValue*. En el proyecto MDHT, en el valor por defecto del atributo se guarda el valor de *mnenomic*, mientras que en los demás sólo se guarda el de *domainName*.

En cuanto a nuestra elección, está claro que los atributos de HL7 se deben transformar a atributos UML. En cuanto a incluir los atributos *typeCode* y *classCode*, se decidió añadirlos en los esquemas resultantes, por la sencilla razón de que si bien es cierto que pueden deducirse, también lo es que no resulta una tarea trivial, puesto que la información que hace referencia a este tema en los *ballots* es compleja y extensa.

Lo que sí se ha considerado oportuno, es indicar que ambos atributos son de sólo lectura utilizando el atributo *isReadOnly* definido en la clase *Property* del metamodelo de UML, ya que su valor está definido y no puede cambiar. Por esta razón, también se ha optado por asignarle el valor por defecto que le corresponde, haciendo uso de la asociación entre *Property* y *ValueSpecification* que existe en el metamodelo de UML para estos casos.

Por otro lado, en cuanto a las propiedades relativas al vocabulario, hemos decidido guardarlas todas en el valor por defecto del atributo, para no perder información. El formato que hemos seguido, se parece bastante al que se emplea en los ficheros HTML relativos a los HMD del estándar. Concretamente, el valor por defecto del atributo tomará por valor: *domainName* + ' , ' + *codeSysteName* + ' : ' + *mnemonic* + ' , ' + 'default = ' + *defaultValue*, donde el símbolo '+', expresa la operación de concatenación de *strings*.

16.2.3 Asociaciones

Las propuestas de todos los proyectos mencionados coinciden en que las asociaciones de HL7, pueden transformarse directamente en asociaciones UML. La única diferencia, es que en la que se hace en la wiki de HL7, se diferencian las asociaciones entre clases de tipo *Role* y *Entity* de tipo *scoper*, de las de tipo *player*, estereotipandolas tal y como se puede apreciar en la figura 16.6.

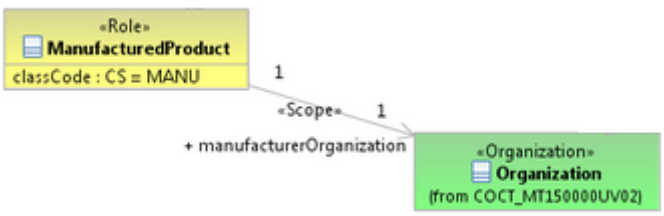


Figura 16.6 Asociación Scoper estereotipada, obtenida de [hl7Wiki]

Está claro que las asociaciones de HL7, deben transformarse en asociaciones UML. Por otro lado, en cuanto a la distinción *scoper*/*player*, no creemos que sea necesario estereotipar las asociaciones para diferenciar los dos tipos.

La forma natural de hacer esta distinción en UML, sería mediante el nombre de rol, y de hecho, muchos de los nombres de rol que aparecen en los esquemas conceptuales del estándar y que se conservan tras la conversión a UML, ya indican claramente si el extremo de la asociación juega el papel de *scoper* o *player*. Es cierto que existen algunos casos donde a partir del nombre de rol no puede deducirse, pero aun así, resulta sencillo saber cuál de los dos papeles se juega mirando el nombre de las clases de tipo *Role* y *Entity* que se relacionan. Además, en los ficheros XSD del estándar, que son los más utilizados a nivel de implementación, no se hace esta distinción *scoper/player,* por lo que no parece ser un tema demasiado importante.

En este punto, podríamos plantearnos algo similar a lo que ya se pensó para el caso de la diferenciación entre las clases de diferentes tipos. Se podría definir un esquema externo con las clases que aparecen en el RIM y sus asociaciones, para poder expresar las relaciones de tipo *scoper*/*player* como redefiniciones de las que apareciesen en dicho esquema. No obstante, de esta manera, nos volveríamos a encontrar el problema de la introducción de dependencias entre esquemas y por los motivos expuestos en el párrafo anterior, consideramos que no merecía la pena y se desestimó esta posibilidad.

16.2.4 Restricciones textuales

En los esquemas de MDHT y en los de *HyperModel*, no aparecen restricciones textuales. En el que se muestra en la wiki de HL7, sí que aparecen, pero recordemos que sólo es una propuesta que no ha llegado a implementarse.

Como ya se comentó anteriormente, las restricciones textuales que están presentes en los esquemas conceptuales del estándar HL7, no se pueden obtener en un formato procesable. Por tanto, como era de esperar, en ninguno de los proyectos aparecen, salvo en el ejemplo de la wiki, en el que seguramente se haya optado por escribir manualmente las que intervienen en ese caso.

En nuestro caso, se han definido las restricciones textuales en el metamodelo de HL7, de forma que si algún día, desde el estándar se proporcionan en un formato procesable, sería relativamente sencillo incorporarlas a la herramienta de transformación de modelos que se ha

construido. En el escenario actual, tratar las restricciones textuales constituye una tarea inasumible al no aparecer en los ficheros MIF, que constituyen nuestra fuente en el proceso de transformación.

16.2.5 Notas

En MDHT y en el ejemplo de la web de *HyperModel* no aparecen notas. Por el contrario, en la versión obtenida tras ejecutar *HyperModel* y en el ejemplo de la wiki de HL7 sí que aparecen y además, en este último caso, se diferencian los dos tipos (descriptivas y de uso) mediante el uso de estereotipos. En ambos casos, el elemento UML utilizado para representar las notas es el comentario (*Comment*).

En un principio tuvimos dudas acerca de sí incluir las notas o no, debido a que cuando un esquema presenta notas descriptivas, casi por cada clase y atributo presente en dicho esquema existe una nota de ese tipo. Pensamos que esto podría suponer un problema al cargar los ficheros UML resultantes en herramientas que permitiesen generar diagramas en formato gráfico a partir de esos ficheros, porque las notas inundarían el diagrama dificultando seriamente su comprensión.

No obstante, finalmente se decidió incluirlas para evitar perder más información de la necesaria en las conversiones realizadas por nuestra herramienta y esperando que esas herramientas de modelado gráfico presentasen opciones para no mostrar las notas por defecto u ocultarlas. De este modo, podríamos asegurarnos de que al menos por nuestra parte, no se ha perdido información y los UML resultantes son correctos.

En UML, el elemento que se utiliza con la misma finalidad con la que se usan las notas en HL7, es el comentario (*Comment*). Por lo tanto, en nuestro caso, por cada nota que exista en un esquema HL7, se creará un comentario en su esquema UML correspondiente.

Una vez se ha decidido que la inclusión de las notas es necesaria, nos tenemos que plantear cómo distinguir las de uso de las descriptivas. En un principio se podría pensar que la solución propuesta en la wiki de HL7, consistente en el uso de estereotipos es la más conveniente, pero lo cierto es que es incorrecta, ya que según lo definido en el metamodelo de UML, no está permitido aplicar estereotipos en los comentarios.

Descartado el uso de estereotipos, se buscó otra manera de distinguir los comentarios, puesto que creíamos que era algo necesario, debido a que las notas de uso y las descriptivas presentan contenidos totalmente diferentes.

Al final, se ha optado por incluir al principio del texto del comentario el *string* '*<Usage>*' o '*<Description>*' según aplique. De este modo, se respetan las reglas del metamodelo de UML y además, se pueden distinguir claramente los dos tipos de notas presentes en los esquemas de HL7.

16.2.6 Choices

En los proyectos mencionados, se pueden encontrar dos maneras de transformar los *choices*. La primera de ellas, empleada en el proyecto MDHT, consiste en crear una clase abstracta con el nombre del *choice* de la cual heredan todas las clases que éste contiene. Las asociaciones en las que participa el *choice* en el esquema HL7, se replican en UML para cada una de las subclases, mientras que las que sólo afectan a una de ellas, siguen afectando únicamente a la subclase que corresponde. De este modo, un *choice* como el que se muestra en la figura 16.7 extraído del esquema *R_Guarantor (universal)* y con identificador *COCT_RM670000UV*, se transformaría en el fragmento UML que se puede apreciar en la figura 16.8. En ese esquema, se muestran aspectos de los cuales aún no se ha hablado, pero por ahora sólo nos interesa fijarnos en la jerarquía que define el *choice* y las relaciones en las que participan él y los elementos que contiene.

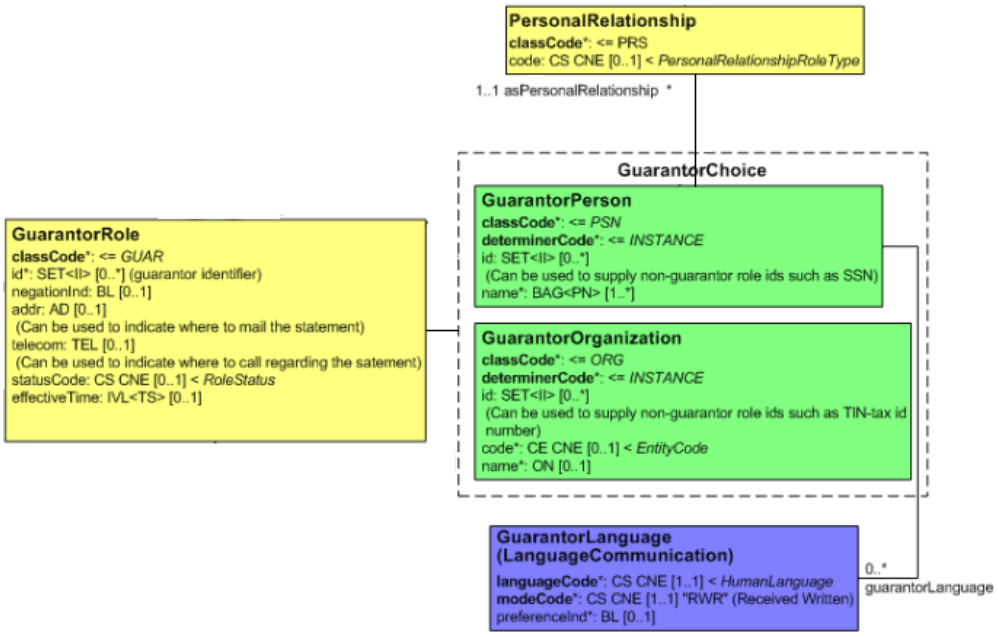


Figura 16.7 Fragmento del esquema R_guarantor universal, obtenido de [Hl7Ball]

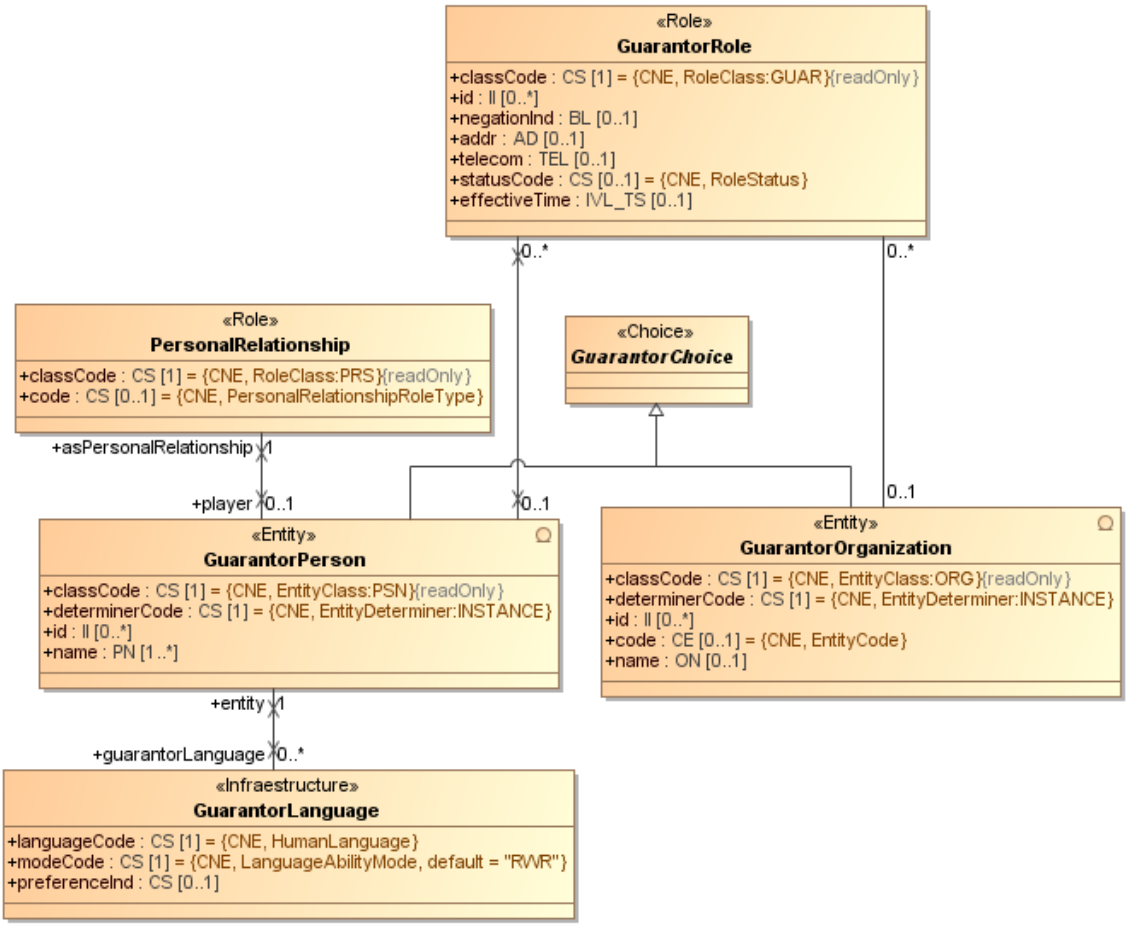


Figura 16.8 Fragmento del esquema R_guarantor universal en UML convertido con MDHT

En los otros tres proyectos, la transformación se realiza de la misma manera, pero sin replicar las asociaciones, es decir, si en el esquema original de HL7, existe una asociación cuyo uno de sus extremos es un *choice*, en UML no se replica dicha asociación para cada una de las subclases de la clase que representa al *choice*, sino que directamente se relaciona el *choice* con el otro extremo de la asociación. Así, el *choice* mostrado en la figura 16.7, se transformaría en el fragmento UML que aparece en la figura 16.9.

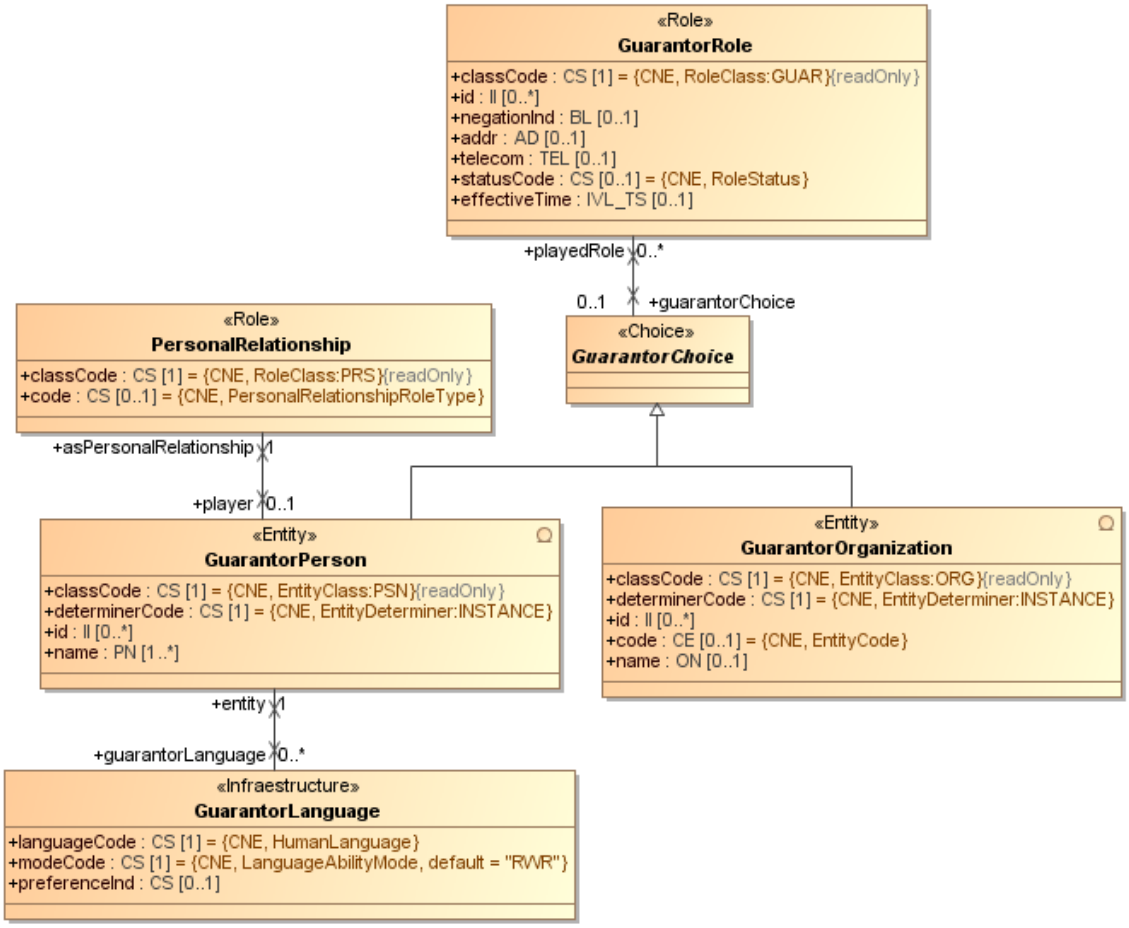


Figura 16.9 Fragmento del esquema R_guarantor universal en UML convertido según las conversiones propuestas desde la wiki de HL7 e HyperModel

Por otro lado, tanto en la wiki de HL7 como en el ejemplo de la web de *HyperModel*, se ha optado por estereotipar la clase abstracta que representa al *choice*, de cara a diferenciarlo de las demás clases.

Por nuestra parte, creemos que el hecho de representar el *choice* mediante una clase abstracta y las clases que contiene mediante subclases en UML, hace que se conserve su semántica y por tanto, constituye una opción correcta.

Nos hemos decantado por la propuesta que se hace desde *HyperModel* y la wiki de HL7, ya que teniendo una jerarquía donde la clase padre representa al *choice*, no le vemos sentido al hecho de replicar las asociaciones tal y como lo hace MDHT. Además, nos hemos decantado por estereotipar la clase que representa al *choice*, porque creemos que es un método apropiado para diferenciarlo del resto de clases.

16.2.7 CMETs

En cuanto a MDHT, en el único esquema que convierten, el *Clinical Document Architecture*, no aparece ningún CMET y por tanto no podemos saber cómo los transformarían.

Por otro lado, desde la wiki de HL7, lo que se propone es crear una clase nueva con el nombre del elemento principal del CMET, con su mismo tipo e indicando el identificador del CMET del que proviene. Cabe destacar, que esta clase no es la misma que contiene dicho CMET, es decir, no se importa, sino que se crea una nueva. En la figura 16.10, se puede observar el resultado de transformar el CMET que aparece en el esquema HL7 de la figura 16.11 según el método que se propone desde la wiki.

Figura 16.10 Conversión a UML del CMET E_Organization universal, obtenida de [hl7Wiki]

Figura 16.11 CMET E_Organization universal, obtenido de [Hl7Ball]

Con lo que respecta a *HyperModel*, esta vez, la propuesta de transformación de CMETs, que se encuentra en el ejemplo de la web, coincide con el resultado que se puede obtener al ejecutar la aplicación. En el UML resultante, en lugar de aparecer una clase representando al CMET, se importa la clase principal del mismo y se la relaciona con las clases con las que el CMET aparece relacionado en el esquema HL7. De este modo, el CMET del esquema de la figura 16.11 mostrada anteriormente, según su propuesta, en UML se representaría tal y como se puede ver en la figura 16.12.

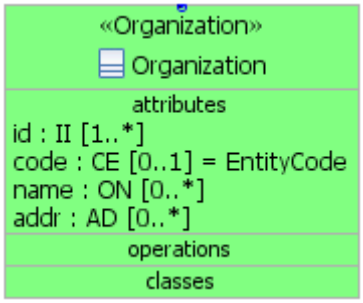


Figura 16.12 Conversión a UML del CMET E_Organization universal, obtenida de [HyperMw]

En cuanto a nuestra elección, creemos que representar los CMETs mediante una clase UML que fuese de tipo CMET y contuviese la misma información que en el esquema HL7 original, sería un error. Puesto que lo que estaríamos haciendo en realidad sería representar un concepto que en la realidad no existe, únicamente por comodidad de cara a la implementación y por proximidad a lo definido en el estándar HL7. Por este motivo, creemos que la propuesta ofrecida en la wiki de HL7 no es correcta.

Bajo nuestro punto de vista, la propuesta de *HyperModel* resulta idónea y por tanto la hemos adoptado. Importando el elemento del CMET pertinente y manteniendo sus asociaciones se consigue preservar la semántica del CMET, cuya función en los esquemas del estándar consiste únicamente en referenciar elementos que se encuentran definidos en otros esquemas.

16.2.8 Entry Point

En el esquema de MDHT y en el que hay disponible en la web de *HyperModel*, no aparecen *Entry Points*. En los dos otros proyectos, en cambio, aparecen representados utilizando el elemento *Interface* de UML. Así, el resultado de transformar el *Entry Point* que aparece en la figura 16.13, extraído del esquema *R_Specimen minimal* con identificador *COCT_RM080100UV*, según estas dos propuestas, sería el mostrado en la figura 16.14.

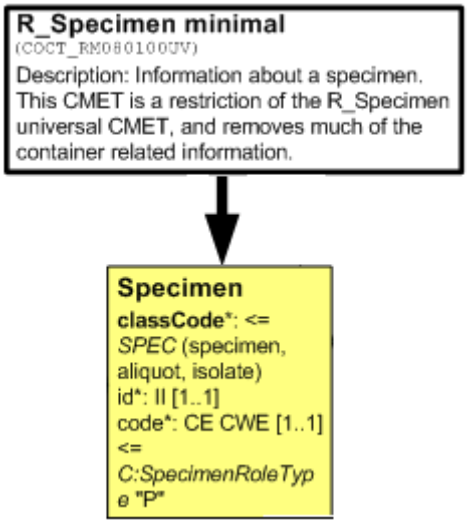


Figura 16.13 Entry Point del esquema R_Specimen minimal, obtenido de [Hl7Ball]

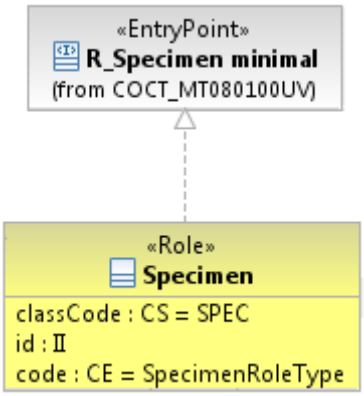


Figura 16.14 Conversión a UML del Entry Point del esquema R_Specimen minimal según HyperModel y MDHT, obtenida de [HyperMw]

Creemos que no representar los *Entry Points* es un error puesto que se pierde información relevante del esquema original, en concreto, su descripción y la distinción de la clase foco del esquema.

En cuanto al uso de *interfaces* que se propone en dos de los proyectos, creemos que no es una opción acertada, puesto que según la definición de *interface* que se puede encontrar en la especificación del metamodelo de UML, ésta se utiliza para definir toda una serie de restricciones que deben cumplir todos los elementos que la implementan. Este uso no corresponde con la semántica definida por los *Entry Points* en los esquemas HL7, donde simplemente se utilizan para señalar la clase foco del diagrama e indicar un par de atributos generales sobre ese esquema, el identificador y descripción.

En nuestro caso, nos hemos decantado por crear una nueva clase estereotipada, que contiene toda la información representada en los *Entry Points* de los esquemas HL7 y relacionarla con la clase foco del esquema original. Esta clase además, es abstracta puesto que no tiene sentido que tenga instancias.

16.2.9 Tipos de datos

En *HyperModel*, los tipos de datos no están implementados, en las transformaciones realizadas por el software sólo aparecen sus multiplicidades, pero no su tipo.

En contraste, tanto en el ejemplo que aparece en su web, como en la herramienta MDHT, se utilizan los mismos tipos de datos que se encuentran definidos en el estándar HL7, pero se suprimen todos los tipos relativos a colecciones, indicando que la multiplicidad del atributo pasa a ser 0..*. Además, cabe destacar que se usan los tipos de datos definidos los archivos XSD del estándar disponibles en [TDBall], es decir, en lugar de usar *IVL<TS>* se utiliza *IVL_TS*, por ejemplo. Esto es debido a que seguramente, sus desarrolladores se toparon con las mismas dificultades descritas en el capítulo 18, dedicado a los problemas que presenta el estándar HL7.

Por último, en la wiki de HL7, se mantienen los tipos de datos que aparecen en el estándar tal cual. En este caso cabe recordar que se trata de una solución que no se ha implementado, seguramente, al llevarla a la práctica, se encontrarían con las dificultades mencionadas en el capítulo dedicado a los problemas que presenta el estándar HL7, y se acabaría por usar los mismos mecanismos de transformación que en los dos proyectos anteriores.

Finalmente, nos decantamos por utilizar el mismo método que se aplica en MDHT y en la web de *HyperModel*. Es decir, utilizar los tipos de datos que se definen en los archivo XSD del estándar, que es lo que al final se acaba utilizando en la práctica, en lugar de la especificación que se puede encontrar en los *ballots*. Además, los tipos que representan colecciones, tampoco los utilizamos, puesto que en la práctica, lo que ellos desde el estándar se define como una colección de un determinado tipo, es equivalente a indicar que el atributo es de dicho tipo y que tienen una cardinalidad 0..*. Una justificación más detallada, se puede encontrar en el capítulo 18 citado anteriormente.

16.2.10 Clases repetidas

Recordemos que en los esquemas del estándar HL7, en ocasiones, se repiten clases de cara a facilitar la comprensión.

Tanto *HyperModel* como MDHT, optan por que todas las clases aparezcan sólo una vez, mientras que en la propuesta que se puede encontrar en la wiki de HL7, dichas clases, aparecen repetidas pero se representan con un color más oscuro y sin atributos, tal y como se puede ver en la figura 16.15, donde la clase *SpecimenInContainer* aparece con un color claramente diferenciado de las otras de tipo *Role*.

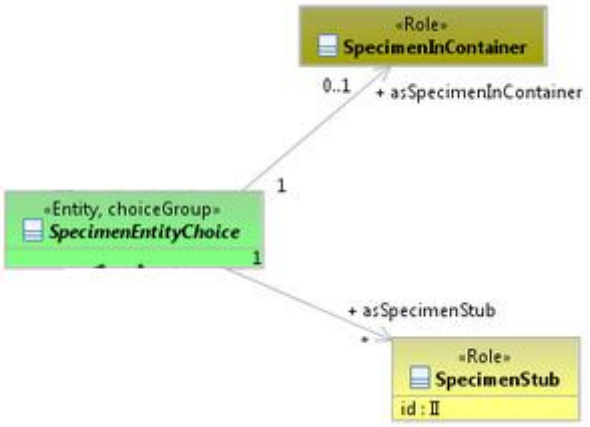


Figura 16.15 Conversión a UML de las clases repetidas, obtenida de [hl7Wiki]

Creemos que la solución propuesta en la wiki de HL7 no es muy acertada, puesto que en realidad lo que se está haciendo es adaptar UML a HL7 y no al revés, tal y como sería de esperar. Por ello, creemos que lo mejor es representarlas como clases normales, sin hacer distinciones por el hecho que aparezcan repetidas en los esquemas originales.

Se podría optar por repetir las clases si se cree que así se facilita la comprensión del esquema. En ese caso, la corrección del esquema no se vería atacada, puesto que aunque se repitan clases, el conocimiento contenido en el esquema conceptual no varía.

No obstante, bajo nuestro punto de vista, en el caso de HL7 no es necesario, puesto que en la mayoría de casos, el número de elementos que aparecen en un fichero MIF no es tan elevado como para que el uso de la técnica de crear réplicas se muestre eficaz. Evidentemente, esta es una afirmación totalmente subjetiva ya que existen tanto partidarios como detractores de dicha técnica.

16.2.11 Conclusiones

Tal y como se ha podido comprobar, las herramientas que existen actualmente y que permiten transformar los diagramas HL7 a UML, se encuentran en un estado de desarrollo más bien temprano y además, resultan muy poco intuitivas para usuarios inexpertos.

MDHT, únicamente convierte uno de los numerosos esquemas presentes en el estándar. Desde la wiki HL7, se hace una propuesta que aún no ha sido implementada, y el proyecto *HyperModel* parece ser que ha sido dejado de lado, no llegando a implementar muchos de los aspectos que se pueden encontrar en el ejemplo de su página web.

Además, como se ha podido comprobar en nuestro análisis, algunas de ellas proponen soluciones que no son correctas o simplemente, no implementan las transformaciones de algunos de los elementos de HL7.

A partir de los motivos expuestos en esta sección, creemos que no resulta difícil justificar la necesidad de proponer alternativas a todos esos proyectos y desarrollar una herramienta como la que se ha construido en el presente proyecto. El objetivo, es cubrir todas las carencias mencionadas, especialmente, la de ofrecer un software que sea capaz de tratar todos los esquemas conceptuales disponibles en el estándar, que sea usable y que realice las conversiones sin errores.

16.3 Definición de las reglas de transformación ATL

Hasta ahora, se han explicado los detalles más importantes de la herramienta ATL y cómo se ha decidido transformar cada uno de los elementos del metamodelo de HL7 a UML. En esta sección, se presentan las reglas de transformación ATL.

El haber expuesto anteriormente las características del lenguaje utilizado por la herramienta y explicado las conversiones que se pretenden implementar, facilitará la comprensión de dichas reglas.

Para empezar, cabe destacar que se ha creado un fichero UML que contiene todos los tipos de datos tal y como se definirán en la sección 18.1, con los atributos que contienen y la jerarquía definida. Esto es, debido a que queremos crear todos los tipos de datos una sola vez y no una por cada transformación que se ejecute.

Con el *profile* que contiene los estereotipos que se desean aplicar, ocurre lo mismo. Queremos crearlos una sola vez y por tanto los definimos en un fichero UML por separado. En concreto, los estereotipos que contiene son los siguientes: *Entry Point*, *Act*, *ActRelationship*, *Participation*, *Role*, *RoleLink*, *Entity*, *Infraestructure*, *cmet* y *Choice*.

De este modo, la cabecera del módulo ATL que define las reglas en nuestro caso, es la que se puede ver en el ejemplo 16.1.

```
module HL7ToXMI;
create OUT : UML2 from IN : HL7, DT : UML2, PRO : UML2;
```

Ejemplo 16.1 Cabecera del módulo ATL de conversión de HL7 a UML

Como se puede observar, se encuentran definidos tres modelos de entrada y uno de salida:

- OUT: modelo UML resultado de ejecutar la transformación.
- IN: metamodelo de HL7.
- DT: modelo en el que se encuentran definidos los tipos de datos.
- PRO: modelos en el que se encuentra definido el *profile* que contiene todos los estereotipos que queremos aplicar.

Antes de comenzar a exponer todas las reglas definidas, cabe destacar que todas ellas son del tipo *called rule*, excepto la que nos conviene que se ejecute primero, la que se trata al procesar el *Entry Point* del esquema, que es de tipo *matched rule*.

El motivo es simple, cada vez que se quiere transformar por ejemplo un *atributo*, necesitamos saber cuál es la clase que lo contiene, para poder asignarla correctamente en el modelo UML resultante. Esto con una *matched rule* no sería posible, ya que en el momento de ejecutarse, no se dispondría de la información de la clase. En cambio, mediante el uso de *called rules* sí que lo es, porque podemos pasar esa información necesaria por parámetro.

Hecha esta aclaración, se procede a mostrar las reglas de transformación que se han definido. Se puede encontrar una regla para cada uno de los elementos HL7 cuya transformación a UML fue explicada en la sección anterior, excepto para los tipos de datos, que como se ha explicado se han definido en un fichero aparte y para las restricciones textuales, que por los motivos anteriormente expuestos, no resulta viable tratarlas. El módulo ATL que contiene todas las transformaciones se puede encontrar en el anexo D.

16.3.1 Entry Point

La regla que se ejecuta al encontrar el *Entry Point* es la primera que se procesa. Es la más extensa de todas, puesto que desde ella se crean todos los elementos iniciales y además, se invoca al resto. Dicha regla, se puede encontrar en el ejemplo 16.2 y se explica a continuación:

- Línea 3: se define el elemento para el cual hay que generar algo en el modelo de salida, que en este caso es el Entry Point.
- Línea 5: se crea un elemento UML de tipo *Model*, que es el que contiene el resto de elementos que se crean.
- Línea 9: se crea el elemento *packageImport* que hace que el modelo anterior, pueda importar el *package* donde están definidos los tipos de datos.
- Línea 14: se crea la clase abstracta que representa al *Entry Point*.
- Línea 18: se crea y se da valor al atributo *description* del *Entry Point*.
- Líneas 28 y 34: se crean las *properties* de los extremos de la asociación entre el Entry Point y la clase foco del esquema.
- Línea 38: se crea la asociación mencionada anteriormente.
- Líneas 42 y 43: se asigna al *Entry Point* su nombre. En algunos esquemas HL7, no aparece el nombre, en estos casos, le asignaremos el identificador en lugar del nombre.
- Línea 44: se aplica el *profile* definido anteriormente.
- Líneas 46 y 47: el *Entry Point* y su asociación con la clase foco se añaden al elemento de tipo *Model*.
- Línea 48: se aplica el estereotipo *EntryPoint* a la clase que representa el *Entry Point*.
- Líneas 50, 53, 56 y 59: se encuentran instrucciones iterativas para llamar por cada clase, cmet, choice y asociación que exista en el modelo de origen, a las *called rules* que tratan esos elementos.
- Línea 62: se asigna el tipo al extremo de la asociación entre el *Entry Point* y la clase foco, en la parte de esta última. No se podía hacer antes, puesto que aún no se habían creado las clases.

```
1. rule EntryPoint {
2.   from
3.     ep: HL7!EntryPoint
4.   to
5.     m: UML2!Model (
6.       name <- ep.identifier,
7.       packageImport <- pi
8.     ),
```

```
9.      pi: UML2!PackageImport (
10.       importedPackage <-
11.        UML2!Package.allInstancesFrom('DT') ->select(p | p.name =
12.        'DataTypes').first()
13.     ),
14.     c: UML2!Class (
15.       ownedAttribute <- pr1,
16.       isAbstract <- true
17.     ),
18.     pr1: UML2!Property (
19.       name <- 'description',
20.       upper <- 1,
21.       lower <- 0,
22.       isReadOnly <- true,
23.       default <- ep.description
24.       type <-
25.         UML2!PrimitiveType.allInstancesFrom('DT')
26.         ->select(pt | pt.name = 'String').first();
27.     ),
28.     pr2: UML2!Property(
29.       upper <- 1,
30.       lower <- 0,
31.       type <- c,
32.       name <- c.name
33.     ),
34.     pr3: UML2!Property(
35.       upper <- -1,
36.       lower <- 0
37.     ),
38.     a: UML2!Association (
39.       ownedEnd <- Set{pr2, pr3}
40.     )
41.   do {
42.     if (not ep.name.oclIsUndefined()) c.name <- ep.name;
43.     else c.name <- ep.identifier;
44.     m.applyProfile(UML2!Profile.allInstancesFrom('PRO')->
45.       select(p | p.name = 'Profile').first());
46.     m.packagedElement.add(c);
47.     m.packagedElement.add(a);
48.     c.applyStereotype(c.getApplicableStereotype(
49.       'Profile::EntryPoint' ) );
50.     for (c in HL7!Class.allInstancesFrom('IN')) {
51.       thisModule.Class(m, c);
52.     }
53.     for (cm in HL7!CMET.allInstancesFrom('IN')) {
54.       thisModule.CMET(m, cm);
55.     }
56.     for (ch in HL7!Choice.allInstancesFrom('IN')) {
57.       thisModule.Choice(m, ch);
58.     }
59.     for (a in HL7!Association.allInstancesFrom('IN')) {
60.       thisModule.Association(m, a);
61.     }
62.     pr3.type <- UML2!Class.allInstancesFrom('OUT')->select(c |
63.       c.name = ep.choosableElement.name).first();
64.   }
65. }
```

Ejemplo 16.2 Regla de transformación ATL de los entry points

16.3.2 Clases

La regla que define qué crear para cada una de las clases existentes en el modelo de HL7 fuente, se llama desde la regla que trata los *Entry Point* y recibe como parámetros el elemento principal de tipo *Model* creado en esa última regla, y una instancia de clase HL7. La regla se muestra en el ejemplo 16.3 y se explica a continuación:

- Línea 3: se crea una instancia de clase UML en el modelo destino y se le asigna un nombre.
- Línea 7: se añade la clase creada al elemento *Model* principal.
- Línea 8: si la clase es de tipo *Act*, se le aplica el estereotipo *Act*.
- Línea 12: lo mismo que en el caso anterior, pero con *ActRelationship*.
- Línea 16: se omiten todas las estructuras condicionales existentes para los cinco tipos de clases restantes, ya que se definen de forma análoga.
- Línea 17: se llama a la *called rule* que trata los atributos por cada uno de los que contiene la clase HL7 fuente.
- Línea 20: se llama a la *called rule* que trata las notas de las clases por cada una de las que contiene la clase HL7 fuente.

```
1. rule Class(m: UML2!Model, c1: HL7!Class) {
2.   to
3.     c2: UML2!Class (
4.       name <- c1.name
5.     )
6.   do {
7.     m.packagedElement.add(c2);
8.     if (c1.oclIsTypeOf(HL7!ActType)) {
9.       c2.applyStereotype(
10.        c2.getApplicableStereotype('Profile::Act'));
11.    }
12.    else if (c1.oclIsTypeOf(HL7!ActRelationshipType)) {
13.      c2.applyStereotype(
14.        c2.getApplicableStereotype('Profile::ActRelationship'));
15.    }
16.    …
17.    for (a in c1.ownedAttribute) {
18.      thisModule.Attribute(c2, a);
19.    }
20.    for (n in c1.note) {
21.      thisModule.ClassNote(c2, n);
22.    }
23.  }
24.}
```

Ejemplo 16.3 Regla de transformación ATL de las clases

16.3.3 Atributos

La regla que crea los atributos en el modelo UML destino, se llama desde la regla que trata las clases y recibe por parámetro tanto un atributo de HL7, como la clase UML que debe contener el atributo que se creará. Dicha regla se encuentra en el ejemplo 16.4 y a continuación, se explican sus detalles más relevantes:

- Línea 3: se crea la *Property* UML que representa al atributo.
- Línea 11: si el atributo es *classCode* o *typeCode*, debe ser de sólo lectura.
- Línea 14: se añade el atributo creado a la lista de atributos de la clase recibida por parámetro.
- Línea 15: para cada una de las notas del atributo, se invoca la *called rule* destinada a la creación de comentarios UML en los atributos.
- Líneas 18 y 21: se tratan las propiedades relativas al vocabulario del atributo, *codingStrength* y *domainName,* en caso que existan.
- Línea 28: se omiten todas las instrucciones que resultan análogas a las dos anteriores y que tratan las tres propiedades restantes del atributo: *codeSystemName*, *mnemonic* y *defaultValue*.
- Línea 29: si en las líneas anteriores se ha obtenido información sobre el vocabulario del atributo porque no era nula, se asigna a su propiedad *default* esa información, representada entre '{' y '}'.

```
1. rule Attribute (c: UML2!Class, p: HL7!Property) {
2.   to
3.     pr: UML2!Property (
4.       name <- p.name,
5.       upper <- p.upper,
6.       lower <- p.lower,
7.       type <- UML2!DataType.allInstancesFrom('DT')->
8.         select(dt | dt.name = p.type.name).first()
9.     )
10.  do {
11.    if (p.name = 'classCode' or p.name = 'typeCode') {
12.      pr.isReadOnly <- true;
13.    }
14.    c.ownedAttribute.add(pr);
15.    for (n in p.note) {
16.      thisModule.AttributeNote(pr, n);
17.    }
18.    if (not p.codingStrength.oclIsUndefined()) {
19.      pr.default <- p.codingStrength;
20.    }
21.    if (not p.domainName.oclIsUndefined()) {
22.      if (not p.codingStrength.oclIsUndefined()) {
23.        pr.default <-
24.          pr.default.concat(',').concat(p.domainName);
```

```
25.       }
26.       else pr.default <- p.domainName;
27.     }
28.     …
29.     if (not pr.default.oclIsUndefined()) {
30.       pr.default <- '{' + pr.default + '}';
31.     }
32.   }
33.}
```

Ejemplo 16.4 Regla de transformación ATL de los atributos

16.3.4 Notas de las clases

La regla que crea los comentarios UML referentes a clases en el modelo destino, se llama una vez por cada nota que haga referencia a una clase HL7. Esta regla presenta como parámetros la clase UML a la que debe hacer referencia el comentario UML que tiene que ser creado, y la nota HL7 de la cual proviene. Esta regla se muestra en el ejemplo 16.5 y a continuación se explica lo más relevante:

- Línea 3: se crea el comentario UML.
- Líneas 5 y 8: se asigna el valor al cuerpo del comentario. Éste consta del cuerpo de la nota HL7, precedida por *<Usage>* o *<Definition>* según su tipo.
- Línea 11: se añade el comentario a la lista de comentarios perteneciente a la clase recibida por parámetro.

```
1. rule ClassNote(cl: UML2!Class, n: HL7!Note) {
2.   to
3.     co: UML2!Comment
4.   do {
5.     if (n.type = 'Usage') {
6.       co.body <- '<Usage> ' + n.body;
7.     }
8.     else if (n.type = 'Definition') {
9.       co.body <- '<Definition> ' + n.body;
10.    }
11.    cl.ownedComment.add(co);
12.  }
13.}
```

Ejemplo 16.5 Regla de transformación ATL de las notas de las clases, CMETs y choices

16.3.5 Notas de los atributos

Esta regla es prácticamente idéntica a la anterior, simplemente cambia el hecho que en lugar de crear comentarios UML que hacen referencia a clases, los que produce, hacen referencia a atributos. La definición de dicha regla, se puede encontrar en el ejemplo 16.6.

```
1. rule AttributeNote(pr: UML2!Property, n: HL7!Note) {
2.   to
3.     co: UML2!Comment
4.   do {
5.     if (n.type = 'Usage') {
6.       co.body <- '<Usage> ' + n.body;
7.     }
8.     else if (n.type = 'Definition') {
9.       co.body <- '<Definition> ' + n.body;
10.    }
11.    pr.ownedComment.add(co);
12.  }
13.}
```

Ejemplo 16.6 Regla de transformación ATL de las notas de los atributos

16.3.6 Asociaciones

La regla que trata las asociaciones, se llama desde la que trata el *Entry Point*, para cada una de las asociaciones HL7 existentes en el esquema conceptual fuente, tal y como se ha visto anteriormente, y recibe dos parámetros: el elemento de tipo *Model*, creado por esa última regla, y una asociación HL7. En esta regla, se puede encontrar una sección llamada *using*, que sirve para definir variables que pueden utilizarse dentro del ámbito que define.

La definición de la regla puede encontrarse en el ejemplo 16.7 y a continuación, se exponen sus aspectos más relevantes:

- Líneas 7 y 10: se guardan en variables las instancias de clases que participan en la asociación que se debe crear.
- Líneas 15 y 21: se crean las *properties* de los extremos de la asociación que se deben crear y en su tipo se asignan las clases creadas anteriormente.
- Línea 27: se crea la asociación y se le asignan las *properties* creadas anteriormente.
- Línea 31: se añade la asociación al elemento *Model* principal.

```
1. rule Association(m: UML2!Model, a1: HL7!Association) {
2.   using {
3.     memberNames : Set(String) = a1.memberEnd->collect(me|
4.       me.type.name);
5.     firstMemberName : String = memberNames.first();
6.     secondMemberName : String = memberNames.last();
7.     firstMemberType : UML2!Class =
8.       UML2!Class.allInstancesFrom('OUT')->select(c | c.name =
9.       firstMemberName).first();
10.    secondMemberType : UML2!Class =
11.      UML2!Class.allInstancesFrom('OUT')->select(c | c.name =
12.      secondMemberName).first();
13.  }
14.  to
15.    p1: UML2!Property(
```

```
16.       upper <- a1.memberEnd.first().upper,
17.       lower <- a1.memberEnd.first().lower,
18.       name <- a1.memberEnd.first().name,
19.       type <- firstMemberType
20.     ),
21.     p2: UML2!Property(
22.       upper <- a1.memberEnd.last().upper,
23.       lower <- a1.memberEnd.last().lower,
24.       name <- a1.memberEnd.last().name,
25.       type <- secondMemberType
26.     ),
27.     a2: UML2!Association (
28.       ownedEnd <- Set{p1, p2}
29.     )
30.   do {
31.     m.packagedElement.add(a2);
32.   }
33.}
```

Ejemplo 16.7 Regla de transformación ATL de las asociaciones

16.3.7 Choices

La regla que crea las clases que representan a los *choices* en el modelo UML destino, se llama desde la regla que trata el *Entry Point* y se invoca una vez por cada *choice* existente en el modelo HL7 fuente. Esta regla recibe como parámetros el elemento UML de tipo *Model*, creado en la regla que trata el *Entry Point*, y un elemento *choice* de HL7. Su definición, puede encontrarse en el ejemplo 16.8 y se explica a continuación:

- Línea 3: se crea la clase abstracta que representa al *choice*.
- Línea 8: se añade la clase creada al elemento *Model* principal.
- Línea 10: se aplica el estereotipo de *Choice* a la clase creada.
- Línea 11: para cada una de las notas que hacen referencia al *choice*, se invoca la misma regla que trata las notas de las clases.
- Línea 14: para cada uno de los elementos que contiene el *choice*, se llama a la regla *ChoiceHierarchy*, que es la que permite construir la jerarquía en el fichero UML resultante.

```
1. rule Choice(m: UML2!Model, ch: HL7!Choice) {
2.   to
3.     c1: UML2!Class (
4.       name <- ch.name,
5.       isAbstract <- true
6.     )
7.   do {
8.     m.packagedElement.add(c1);
9.     c1.applyStereotype( c1.getApplicableStereotype(
10.      'Profile::Choice'));
```

```
11.     for (n in ch.note) {
12.       thisModule.ClassNote(c1, n);
13.     }
14.     for (oe in ch.ownedElement) {
15.       thisModule.ChoiceHierarchy(c1,
16.         UML2!Class.allInstancesFrom('OUT')->select(c | c.name =
17.         oe.name).first());
18.     }
19.   }
20.}
```

Ejemplo 16.8 Regla de transformación ATL de los choices

16.3.8 Jerarquía de choices

Esta regla únicamente sirve para que se pueda definir la jerarquía de clases que corresponde al *choice* en el fichero UML destino. Como se ha explicado anteriormente, se llama siempre desde la que trata los *choices* y recibe dos clases UML como parámetro. La regla se encuentra en el ejemplo 16.9 y es muy fácil de entender, simplemente hay que tener en cuenta que hay que crear un objeto UML de tipo *Generalization*, indicando cuál de las dos clases recibidas por parámetro es la general, y cuál la especifica. Una vez creado, este objeto se tiene que añadir a la colección de generalizaciones que contiene la clase específica.

```
1. rule ChoiceHierarchy(parentClass: UML2!Class, childClass:
2.  UML2!Class) {
3.   to
4.     g: UML2!Generalization (
5.       general <- parentClass,
6.       specific <- childClass
7.     )
8.   do {
9.     childClass.generalization.add(g);
10.  }
11.}
```

Ejemplo 16.9 Regla de transformación ATL de la jerarquía de los choices

16.3.9 CMET

Debido a las limitaciones impuestas por la herramienta ATL, no nos resulta posible, de forma directa, generar el modelo UML resultante de la forma como se había decidido, que básicamente consistía en sustituir los CMETs por las clases foco del esquema que definen, que serían importadas por el modelo que incluye el CMET en cuestión.

Para poder llevar a cabo esa tarea, se necesitaría para realizar una sola transformación, abrir todos los esquemas conceptuales disponibles en el estándar HL7, porque a priori, se

desconoce de qué esquemas se necesitaría importar elementos. Además, se necesitaría ir modificando a la vez todos ellos.

Esto último es inasumible porque ATL no nos permite modificar los modelos de entrada. Por ello, se ha optado por tratar este aspecto en un proceso posterior, cuando se disponga de todos los esquemas convertidos. Este aspecto corresponde a la última fase del desarrollo de la herramienta de conversión y se trata con detalle en el siguiente capítulo.

Por el momento, la regla que trata los CMETs, se encarga de generar en el modelo UML destino una clase que contiene toda la información relativa al CMET y que resulta útil de cara a realizar al proceso de refinamiento mencionado en el párrafo anterior. Dicha regla, se encuentra en el ejemplo 16.10 y sus aspectos más relevantes se listan a continuación:

- Línea 3: se crea la clase UML que representa al CMET.
- Línea 7: se crea y se da valor al atributo *identifier* del CMET creado.
- Línea 17: se añade el CMET creado al elemento *Model* principal.
- Línea 18: se aplica el estereotipo CMET a la clase creada.
- Líneas 20 y 24: se aplica un segundo estereotipo a la clase creada dependiendo del tipo del elemento principal del CMET. Para simplificar, sólo se muestran los condicionales que tratan el caso del tipo *Act* y *ActRelationship*.
- Línea 29: para cada una de las notas que hacen referencia al CMET, se invoca a la misma regla que trata las notas de las clases.

```
1. rule CMET(m: UML2!Model, cm: HL7!CMET) {
2.   to
3.     c1: UML2!Class (
4.       name <- cm.name,
5.       ownedAttribute <- pr
6.     ),
7.     pr: UML2!Property (
8.       name <- 'identifier',
9.       upper <- 1,
10.      lower <- 1,
11.      type <- UML2!PrimitiveType.allInstancesFrom('DT')
12.       ->select(pt | pt.name = 'String').first(),
13.      isReadOnly <- true,
14.      default <- cm.identifier
15.    )
16.  do {
17.    m.packagedElement.add(c1);
18.    c1.applyStereotype(
19.      c1.getApplicableStereotype('Profile::cmet'));
20.    if (cm.mainClassType = 'Act') {
21.      c1.applyStereotype(
22.        c1.getApplicableStereotype('Profile::Act'));
23.    }
```

```
24.     else if (cm.mainClassType = 'ActRelationship') {
25.       c1.applyStereotype(
26.         c1.getApplicableStereotype('Profile::ActRelationship'));
27.     }
28.     …
29.     for (n in cm.note) {
30.       thisModule.ClassNote(c1, n);
31.     }
32.   }
33.}
```

Ejemplo 16.10 Regla de transformación ATL de los CMETs

16.4 Resultado de ejemplo

En esta sección, se muestran y se explican los detalles del modelo UML que se obtiene como resultado de ejecutar las reglas ATL para el fichero XML que se obtuvo a partir del MIF del esquema conceptual del estándar con identificador *COCT_MT030000UV04*.

Las explicaciones relativas a dicho esquema y al fichero XML ya fueron explicadas en la sección 14.5. Por este motivo, aquí no se vuelven a repetir, no obstante, creemos conveniente volver a adjuntar el esquema conceptual (figura 16.16) para que resulte más cómodo seguir las explicaciones presentadas a continuación.

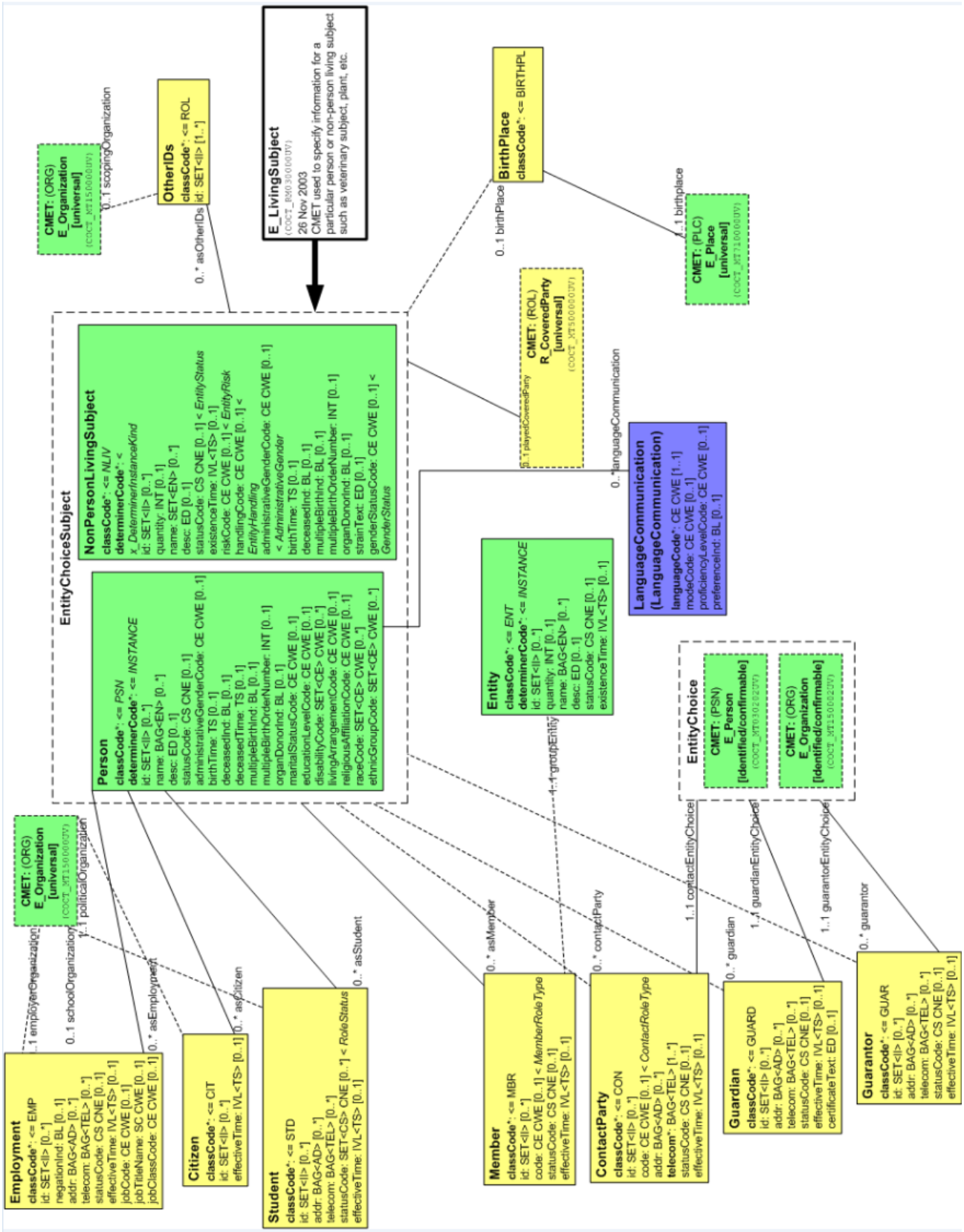


Figura 16.16 Esquema E_LivingSubject, obtenido de [Hl7Ball]

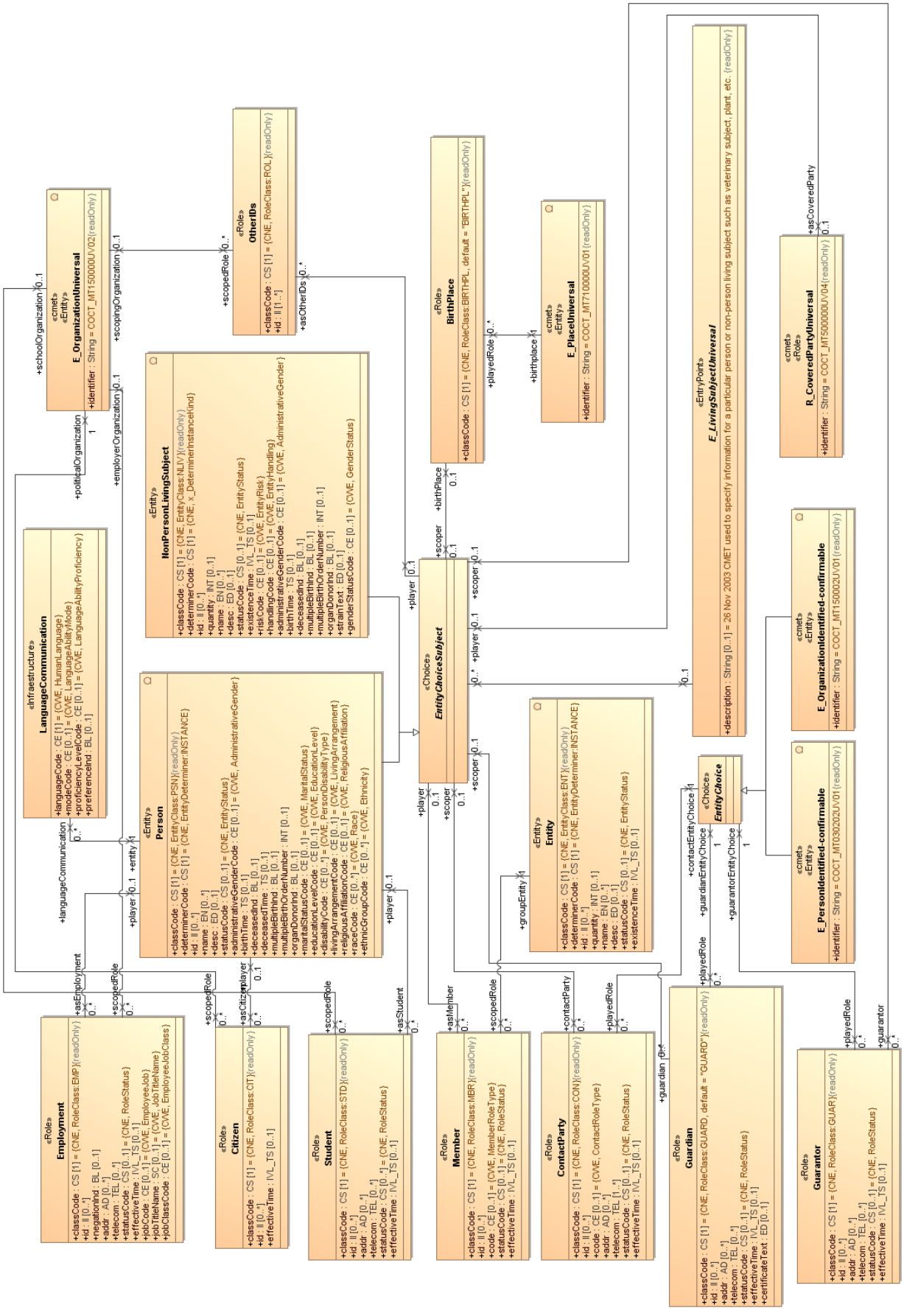


Figura 16.17 Conversión a UML del esquema E_LivingSubject tras ejecutar las reglas ATL

A continuación, se expone con la ayuda de ejemplos extraídos del esquema conceptual UML resultante mostrado, que efectivamente, las reglas ATL definidas, implementan las transformaciones descritas anteriormente.

Primero de todo cabe señalar, que aunque no aparezca explícitamente en el esquema conceptual mostrado, todos los elementos que se pueden ver están contenidos en un elemento de tipo *Model*, cuyo nombre es el identificador del esquema, en este caso *COCT_MT030000UV04*.

Tal y como se puede apreciar, el *Entry Point* que aparece en el esquema HL7 original, se transforma en una clase abstracta, con el estereotipo *EntryPoint* y que contiene un único atributo, la descripción, que como se puede observar, es la misma que aparece en el esquema fuente. Además, está relacionado con la clase foco del esquema, la apuntada por el *Entry Point*, que en este caso es *EntityChoiceSubject*.

En cuanto a las clases, se puede apreciar que por cada una de las existentes en el esquema HL7 de origen, se ha creado una en el esquema UML destino. Además, cada una de ellas aparece estereotipada según su tipo. Así por ejemplo, a la clase Person se le ha aplicado el estereotipo *Entity* y a la clase *Student*, *Role*.

Por lo que respecta a los atributos, al igual que ocurre con las clases, por cada uno existente en el esquema HL7 origen, se crea uno en el esquema UML destino. Además, se puede comprobar que todos los atributos *classCode* y *typeCode* aparecen señalados como de sólo lectura (*readOnly*) y que aquellos que son de tipo código, a su lado contienen entre los caracteres '{' y '}' toda la información relativa a ese código. En cuanto a esto último, es importante señalar que a pesar de que algunos de esos datos no aparezcan en el esquema original, sí que aparecen en el fichero MIF que lo representa.

En lo referente a las asociaciones, vemos que se ha creado una por cada una de las existentes en el esquema original y que conservan sus nombres de rol y multiplicidades. En este aspecto, al igual que ocurre con la información referente a los atributos de tipo código, existen nombres de rol y multiplicidades que están presentes en los ficheros MIF, pero que no aparecen representados en el esquema conceptual gráfico. Así por ejemplo, vemos que en la asociación que une las clases *Person* y *Student*, en el extremo de esta última aparece el nombre de rol *asStudent* y la multiplicidad 0..*, que coincide con lo que se puede ver en el esquema original, pero en el extremo de la clase *Person* del esquema HL7 no aparecen reflejados esos dos datos.

En cuanto a los *choices*, se puede observar que a partir de los dos existentes en el esquema original, que son *EntityChoice* y *EntityChoiceSubject*, en el esquema resultante, se han creado dos clases abstractas con el mismo nombre y con el estereotipo *Choice* aplicado. Además, para cada una de ellas, podemos ver una jerarquía definida, en la que sus subclases, representan los elementos que estos *choices* contienen en el esquema de origen. De este modo, vemos que las clases *Person* y *NonPersonLivingSuject* son subclases de *EntityChoiceSubject* en el esquema UML. Por último, cabe destacar que todas las asociaciones en las que participan los *choices* y sus subclases que aparecen en el esquema original, se mantienen en el resultante. Así, por ejemplo, se puede comprobar que el *choice* con nombre EntityChoiceSubject sigue relacionado con la clase *Birthplace*, del mismo modo que su subclase *Person* mantiene la asociación con la clase *Citizen*.

En lo que se refiere a los CMETs, se puede comprobar que se han transformado en clases con dos estereotipos, *cmet* y el que le corresponda según el tipo de su elemento principal. De este modo, a la clase que representa al CMET *E*_Place, se le han aplicado tanto los estereotipos *cmet* como *Entity.* Además, presentan un atributo de sólo lectura que contiene el identificador del esquema conceptual de HL7 con toda la información de dicho CMET. Tal y como se ha afirmado anteriormente, esta transformación de los CMETs no es definitiva, y se verá modificada en el proceso de refinamiento descrito en el próximo capítulo.

Por otro lado, cabe recordar que aunque se repitan clases o CMETs en el esquema original, éstos no aparecen repetidos en el esquema UML destino. En el ejemplo mostrado, se puede apreciar que en el esquema HL7, el CMET *E_Organization[universal]* aparece dos veces y que, sin embargo, en el esquema UML aparece una sola vez, eso sí, mantiendo todas las asociaciones existentes en el esquema original.

Por último, queda por analizar lo que ocurre con las notas. A partir del ejemplo mostrado, no nos resulta posible comprobar que al ejecutar las transformaciones ATL que convierten las notas de HL7 en comentarios de UML se obtienen los resultados esperados, puesto que este modelo no contiene notas. No obstante, en el capítulo 19 se puede ver un ejemplo de transformación que sí que contiene.

16.4.1 XMI

En realidad, lo que generan las transformaciones ATL, no es un gráfico como el mostrado en la figura 16.17, sino que es un fichero XMI textual, que recordemos que básicamente es un esquema UML representado en un formato procesable.

Esto muestra una de las ventajas de utilizar UML. Teniendo el fichero XMI, se ha podido utilizar una herramienta externa, en este caso Magic Draw, disponible en [MDraw], que permite generar un esquema conceptual en formato gráfico a partir de él. Además, para acercarnos más al estilo de presentación utilizado en HL7, se podrían cambiar los colores tal y como se muestra en la figura 16.18. No sería complicado crear una herramienta que cargarse esos esquemas y cambiase los colores según el tipo de clase, pues bastaría con fijarse en el estereotipo que se aplica a cada una de las clases.

Los ficheros XMI se pueden tratar con *Eclipse*, que nos los muestra con una vista jerárquica muy sencilla de comprender. En la figura 16.19, se muestra esta vista para el mismo esquema UML ejemplificado.

No se muestran todos los elementos por limitaciones de espacio. No obstante, el fragmento mostrado es perfectamente válido para hacernos una idea de lo que nos ofrece *Eclipse*.

En él, se puede ver como se pueden desplegar los elementos para acceder a sus propiedades internas, así por ejemplo, se muestran todos los atributos de la clase *Person* y para uno de ellos su multiplicidad y valor por defecto.

Desde esta vista, se pueden apreciar todas las características anteriormente mencionadas con respecto al esquema UML gráfico. Vemos por ejemplo, que se aplican los estereotipos *Entry Point*, *Entity* y *Role*, que los atributos contienen sus tipos de datos y cardinalidad, etc.

Evidentemente, este archivo XMI, también puede abrirse con cualquier editor de texto, aunque naturalmente no resulta tan fácil de entender como un esquema gráfico o la vista jerárquica que nos ofrece *Eclipse*. Principalmente porque todos los elementos se referencian con códigos alfanuméricos bastante extensos. Tan sólo para ver su estructura, en el ejemplo 16.11, se muestra la definición en XMI de la clase *Citizen* del esquema mostrado anteriormente en la figura 16.18.

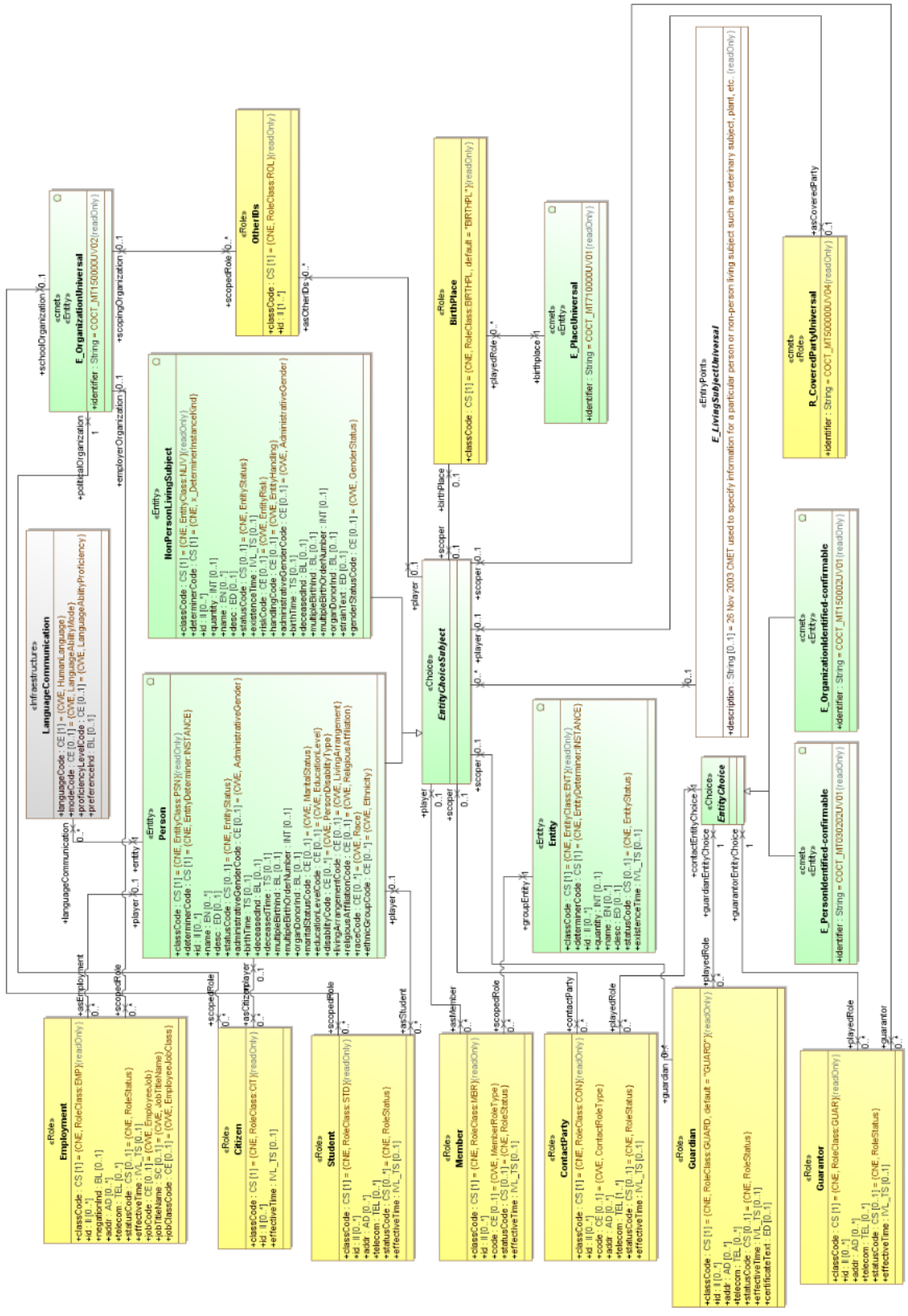


Figura 16.18 Esquema UML con los colores empleados por HL7

Como se puede comprobar, el elemento de tipo *packagedElement* es el que representa a la clase, y presenta como hijos elementos de tipo *ownedAttribute*, que representan sus atributos. Éstos a su vez, presentan como hijos cuatro tipos de elementos: *type*, *upperValue*, *lowerValue* y *defaultValue*. En el primero se define el tipo del atributo, en el segundo y en el tercero su multiplicidad, y en el último su valor por defecto en caso de que tenga.

```
<packagedElement name="Citizen" xmi:id="_qTg1hIIsEeCP6KBOdKlUVg"
 xmi:type="uml:Class">
  <ownedAttribute isReadOnly="true" name="classCode"
   xmi:id="_qTg1hYIsEeCP6KBOdKlUVg">
    <type href="./DataTypes.uml#_HJOJQGv3EeCqNv_5ArIiEQ"
     xmi:type="uml:DataType"/>
    <upperValue value="1" xmi:id="_qTg1hoIsEeCP6KBOdKlUVg"
     xmi:type="uml:LiteralUnlimitedNatural"/>
    <lowerValue value="1" xmi:id="_qTg1h4IsEeCP6KBOdKlUVg"
     xmi:type="uml:LiteralInteger"/>
    <defaultValue value="{CNE, RoleClass:CIT}"
     xmi:id="_qTg1iIIsEeCP6KBOdKlUVg" xmi:type="uml:LiteralString"/>
  </ownedAttribute>
  <ownedAttribute name="id" xmi:id="_qTg1iYIsEeCP6KBOdKlUVg">
    <type href="./DataTypes.uml#_yL3JEGv3EeCqNv_5ArIiEQ"
     xmi:type="uml:DataType"/>
    <upperValue value="*" xmi:id="_qTg1ioIsEeCP6KBOdKlUVg"
     xmi:type="uml:LiteralUnlimitedNatural"/>
    <lowerValue xmi:id="_qTg1i4IsEeCP6KBOdKlUVg"
     xmi:type="uml:LiteralInteger"/>
  </ownedAttribute>
  <ownedAttribute name="effectiveTime"
   xmi:id="_qTg1jIIsEeCP6KBOdKlUVg">
   <type href="./DataTypes.uml#_6soQQGv3EeCqNv_5ArIiEQ"
    xmi:type="uml:DataType"/>
   <upperValue value="1" xmi:id="_qTg1jYIsEeCP6KBOdKlUVg"
    xmi:type="uml:LiteralUnlimitedNatural"/>
   <lowerValue xmi:id="_qTg1joIsEeCP6KBOdKlUVg"
    xmi:type="uml:LiteralInteger"/>
  </ownedAttribute>
</packagedElement>
```

Ejemplo 16.11 Definición en UML de la clase Citizen

Figura 16.19 Captura de la vista jerárquica ofrecida por Eclipse

16.5 Resumen

A lo largo de este capítulo, se han mostrado todos los detalles de la fase del proyecto consistente en la definición de las reglas de transformación.

Recordemos que en el momento de iniciar esta fase del proyecto, ya contábamos con un metamodelo de HL7 definido y con una instancia XML de ese metamodelo por cada uno de los esquemas conceptuales del estándar HL7 que se deseaba transformar. Estos dos elementos, junto al metamodelo UML, que ya se encontraba definido, constituían todos los requisitos indispensables para poder realizar las transformaciones del software ATL.

En este capítulo, se han explicado las propuestas de conversión para cada uno de los elementos de HL7 que se hacen desde las diferentes herramientas existentes, a la vez que se valoraban y se escogía la que creíamos que era más acertada, o proponíamos una nueva en caso que ninguna nos resultase suficientemente convincente.

Posteriormente, se han definido las reglas ATL para llevar a cabo esas transformaciones y finalmente, se han mostrado los detalles más importantes de los esquemas UML resultado de ejecutar esas reglas.

17 REFINAMIENTO DE LOS ESQUEMAS UML

En este capítulo, se describen todos los detalles relacionados con la última fase de la construcción de la herramienta de transformación, el refinamiento de los esquemas UML obtenidos en la fase anterior.

Como ya se ha introducido en el capítulo anterior, los esquemas conceptuales UML resultantes de la aplicación de dichas reglas no cumplen exactamente con todas las transformaciones que se habían propuesto, debido a ciertas limitaciones que impone la herramienta de transformación de modelos ATL.

Los aspectos para los que se debe encontrar solución son dos, y ambos están relacionados con las referencias a esquemas conceptuales externos.

El primer problema es que las referencias a los modelos UML que contienen la información de los tipos de datos y de los estereotipos aplicados, no son correctas.

El segundo es que tal y como se afirmó anteriormente, los CMETs no se han sustituido por sus clases principales como se propuso en la sección dedicada a las propuestas, sino que se optó por una solución alternativa, ya que no era posible llevar a cabo la escogida utilizando ATL.

Estos problemas, han sido solventados desarrollando una aplicación codificada en Java, cuyas entradas son los esquemas conceptuales UML obtenidos en la fase anterior del proyecto, y su salida, esos mismos esquemas UML, pero con los dos problemas anteriormente mencionados corregidos.

A continuación, se presentan de forma detallada ambos problemas, junto a la solución propuesta para solventarlos.

17.1 Referencias a tipos de datos y profile

Tal y como se ha explicado anteriormente, los tipos de datos y el *profile* que contiene todos los estereotipos que se aplican en los esquemas UML resultantes, están definidos en esquemas UML aparte.

El motivo de esto, es que se desea evitar el tener que definirlos en cada uno de los esquemas. Es mejor hacerlo en uno por separado y que todos los demás los referencien.

El referenciarlos, se traduce en introducir una ruta hacia esos esquemas, y el problema, es que la herramienta ATL especifica una ruta que sólo es válida dentro de la plataforma *Eclipse*. De este modo, cada vez que se indica el tipo de un atributo, encontramos lo que se puede apreciar en el ejemplo 17.1.

```
<type xmi:type="uml:DataType"
href="platform:/resource/HL7ToXMI/DataTypes.uml#_HJOJQGv3EeCqNv_5ArIi
EQ"/>
```

Ejemplo 17.1 Referencia a un esquema externo válida únicamente en Eclipse

En este ejemplo, *DataTypes.uml*, es el nombre del esquema que contiene la definición de todos los tipos de dato, mientras que el *string* que aparece tras el carácter '#', es el identificador que ese tipo de datos tiene asignado en el esquema *DataTypes.uml*.

El enlace *platform:/resource/HL7ToXMI/DataTypes.uml*, lo que está referenciando, es el esquema *DataTypes.uml* dentro del proyecto HL7ToXMI de *Eclipse*. Esto, evidentemente funciona correctamente mientras se trabaja con esa plataforma, pero si lo que se desea es que el esquema UML que lo contiene se pueda cargar en otro tipo de herramientas, es necesario cambiar esa ruta.

Desde el módulo que contiene las reglas de transformación ATL, no se puede hacer nada para cambiar el tipo de enlaces generados. Por ello, se hace necesario abordar la problemática descrita en un proceso posterior a la transformación.

En cuanto a la herramienta utilizada para redefinir los enlaces, utiliza las mismas funciones de la API Java que se usaron al tratar los archivos MIF del estándar. De modo que lo único que hay que hacer, es ir recorriendo el fichero UML y sustituir todos esos enlaces por otros que no sean relativos a *Eclipse*.

Así, lo mostrado en el ejemplo anterior tras ejecutar esa herramienta, quedaría tal y como se muestra en el ejemplo 17.2. Esto hará que el esquema que contiene el enlace, se pueda cargar en cualquier herramienta mientras esté en el mismo directorio que el esquema *DataTypes.uml*.

```
<type xmi:type="uml:DataType"/>
href="./DataTypes.uml#_HJOJQGv3EeCqNv_5ArIiEQ"
```

Ejemplo 17.2 Referencia a un esquema externo definida correctamente

Por otro lado, con el *profile* que contiene todos los estereotipos, *hl7.profile.uml*, ocurre exactamente el mismo problema y su solución también pasa por redefinir los enlaces que lo referencian.

17.2 Inclusión de los elementos principales de los CMETs

En los esquemas UML resultantes de las transformaciones realizadas con ATL, los CMETs de HL7 se convierten en clases UML a las que se les aplica el estereotipo *cmet* y el del tipo de su clase principal, y que contienen un atributo que indica el identificador del esquema HL7 del que se puede extraer su información.

Esto era debido a que a causa de las limitaciones impuestas por ATL, no resulta posible que en el esquema UML resultante, en vez de crear una clase con la información relativa al CMET que contiene el esquema fuente, se defina directamente el elemento principal de éste. Este problema también se tiene que corregir, necesariamente, una vez se han ejecutado las reglas de transformación para todos los esquemas que se desean convertir.

Lo que hace la herramienta construida en esta fase del proyecto, para tratar el tema de los CMETs, es borrar todas las clases con estereotipo *cmet* que se habían creado y sustituirlas por la clase principal del CMET al que hacen referencia, teniendo en cuenta que en las asociaciones en las que participaba el CMET, ahora lo hace su clase principal.

Al importar la clase, los enlaces se definen de manera que no se den los problemas de referencias descritos en la sección anterior.

Lo descrito puede hacerse de forma relativamente sencilla. Los problemas surgen cuando los CMET que se desean sustituir están incluidos en un elemento de tipo *choice*.

Esa circunstancia, en UML se traduce en definir un elemento de tipo *Generalization* en el que la clase que representa al *choice* es la general, y la que representa al CMET, la específica. Y *Eclipse* y otras herramientas como *MagicDraw*, siempre incluyen ese elemento *Generalization* en uno de los conjuntos que contiene la clase específica.

Este último hecho, aumenta considerablemente la complejidad de las referencias que deben definirse entre esquemas. A continuación, se detallan los pasos que se deberían tomar en el caso del *choice* mostrado en la figura 17.1, que ha sido extraído del esquema con nombre *E_LivingSubject* e identificador *COCT_MT03000UV*.

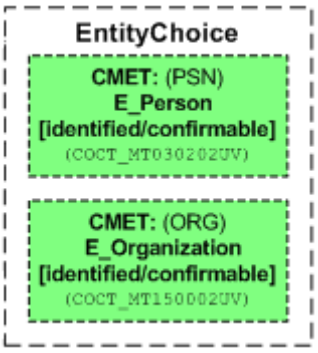


Figura 17.1 Choice EntityChoice, obtenido de [Hl7Ball]

Para el CMET *E_Person*, lo primero que se debería hacer es, tal y como ya se ha explicado anteriormente, definir una clase en el esquema UML, que represente a la clase principal de dicho CMET, que es *Person*. Para ello, simplemente hay que importarla del esquema con identificador *COCT_MT030202UV*.

Acto seguido, se debe crear el elemento de tipo *Generalization* que contenga la información de herencia entre la clase UML que representa al *choice EntityChoice* y la que representa a la clase principal del CMET *E_Person*.

Este elemento *Generalization* debe incluirse dentro de la información perteneciente a la clase hija, lo que significa que en este caso, tiene que estar en la clase *Person* del esquema con identificador *COCT_MT030202UV*, el problema es que en dicho esquema, no se tiene la información de la clase que representa al *choice EntityChoice* del esquema *COCT_MT03000UV*. Por tanto, desde el esquema *COCT_MT0302002UV* debe importarse dicha clase desde el *COCT_MT03000UV*.

En resumen, por lo que respecta a las referencias, lo que se debe hacer es importar desde el esquema con identificador *COCT_MT03000UV* la clase *Person* de *COCT_MT030202UV* y desde este último, la clase *EntityChoice* presente en *COCT_MT03000UV*.

Es cierto que de esta manera se están introduciendo referencias cruzadas entre distintos esquemas UML, sin embargo, esta situación no es evitable. Si bien es cierto que esto no hace que los esquemas sean incorrectos, sí que lo es que puede presentar inconvenientes dependiendo de la herramienta utilizada para cargar y trabajar con esos esquemas.

Así por ejemplo, en *Eclipse* no existe ningún problema, pero *MagicDraw* cuando carga un esquema, sólo carga ese esquema y todos aquellos de los que importa directamente.

Esto es un problema, puesto que imaginemos que en un esquema A, se tiene un *choice* que contiene un CMET. La clase principal de ese CMET, deberá ir a buscarse a un esquema B, pero esa misma clase podría heredar de otra clase definida en un esquema C. Como el esquema C, no depende directamente de A, al cargar este último en la herramienta, no carga C y no muestra todos los elementos que dependen de otros definidos en ese esquema C.

Este hecho, ha provocado que se opte por ofrecer dos versiones de los esquemas UML resultantes. En la primera se resuelven los dos aspectos comentados en este capítulo, mientras que en la segunda, únicamente se redefinen los enlaces que incluyen rutas propias de *Eclipse*.

De este modo, los usuarios que deseen una versión mejor desde el punto de vista de la corrección y que utilicen herramientas que no presenten problemas al cargarlos, utilizarán la primera versión descrita, mientras que aquellos que no dispongan de dichas herramientas, o quieran tener toda la información disponible en un único esquema, podrán optar por utilizar la segunda.

17.3 Resumen

En este capítulo, se han explicado los detalles relativos a la última fase de la construcción de la herramienta de transformación de modelos HL7 a UML, correspondiente al refinamiento de los esquemas UML obtenidos al ejecutar las transformaciones definidas en ATL y expuestas en el capítulo anterior.

Concretamente, se han explicado dos problemas que no se podían resolver utilizando ATL y cómo se han solventado desarrollando una pequeña aplicación escrita en Java.

18 PROBLEMAS DEL ESTÁNDAR HL7

En este capítulo, se analizan todos los inconvenientes con los que nos hemos topado al trabajar con el estándar HL7 y que, inevitablemente, afectan a los resultados producidos por la herramienta que se ha desarrollado en este proyecto y a su proceso de construcción.

18.1 Tipos de datos

Existe una especificación UML para los tipos de datos de HL7, por ejemplo la del *ballot* de septiembre de 2010 se puede encontrar en [TDBall].

El problema es que la información contenida en dicha especificación, no es reflejada fielmente por los ficheros XML que definen los tipos de datos a nivel de implementación. Las diferencias son múltiples y acentuadas puesto que no coinciden algunos nombres de los tipos, ni todos los atributos, ni las relaciones de herencia, etc.

Este hecho nos plantea la disyuntiva de si optar por seguir lo definido en la especificación UML, o basarnos en lo que se puede encontrar en los ficheros XML.

Aunque a priori pueda parecer una decisión errónea, se ha optado por seguir lo definido a nivel de implementación. La justificación es simple, lo que se ha especificado en UML no se está utilizando en el intercambio de mensajes HL7 que se produce entre distintos sistemas reales. Los tipos de datos que se utilizan, obviamente, son los que se encuentran en los ficheros XML del estándar.

Recordemos que uno de los objetivos de este proyecto es que a partir de los esquemas conceptuales UML generados, sea posible utilizar herramientas que trabajen con dicho lenguaje y que se pueda facilitar la labor de aquellas personas que trabajan con el estándar. Anteriormente se ha puesto como ejemplo un software que a partir de UML sea capaz de generar código automáticamente. Obviamente, en este caso, lo que interesa es que el código generado sea compatible con lo que realmente se está utilizando a día de hoy a nivel de implementación. De nada serviría que se siguiese lo definido en una especificación que luego no se utiliza.

Por otro lado, la especificación de los tipos de datos en UML que se proporciona desde el estándar contiene varios errores.

Uno de ellos, es que en ocasiones, la clase más general dentro de una jerarquía, se dota de todos los atributos que pueden contener sus subclases y en éstas, se definen restricciones que indican cuáles de los atributos heredados deben ser nulos.

Es una solución que se distancia de lo que se acepta como correcto en UML, que consiste en tener en la clase más general únicamente los atributos compartidos por todas las subclases, definiendo en cada una de ellas los específicos. En la figura 18.1, se puede apreciar un fragmento de la especificación UML del estándar donde se da la situación descrita.

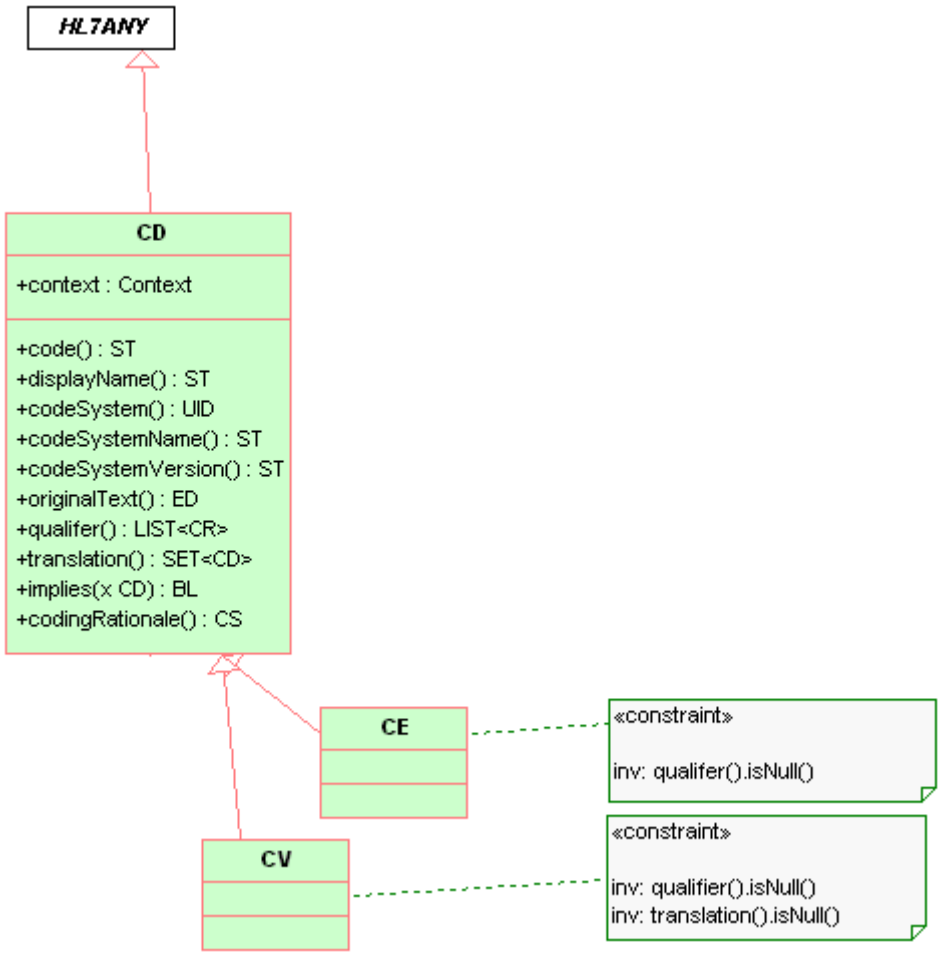


Figura 18.1 Fragmento de la especificación de los tipos de datos, obtenido de [Hl7Ball]

En el fragmento mostrado, el tipo *CE*, contiene una restricción que indica que el atributo *qualifier* debe ser nulo y *CV*, contiene otra, que indica que tanto *qualifier* como *translation* deben serlo. Tal y como se ha comentado anteriormente, una mejor solución consistiría en definir esos dos últimos atributos únicamente en aquellos tipos donde apliquen, en lugar de hacerlo en la clase que representa al tipo más genérico, de esta forma no resultaría necesario el uso de restricciones textuales.

El ejemplo mostrado nos lleva a analizar otro de los problemas que presenta la especificación de los tipos de datos del estándar, que es el hecho que en ocasiones se definen operaciones que retornan atributos que no se han definido dentro de la clase.

Como se puede apreciar, la clase *CD* contiene operaciones que retornan atributos que no están definidos. Así por ejemplo, *displayName(),* retorna un el atributo *displayName* de tipo *string*, pero dicho atributo no se encuentra definido en la clase. Tampoco puede encontrarse en HL7ANY (figura 18.2), que es la clase de la cual hereda y la que se sitúa en el nivel más alto de la jerarquía de los tipos de datos.

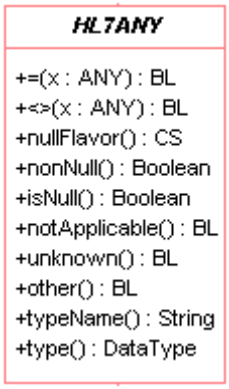


Figura 18.2 Tipo de dato HL7Any, obtenido de [Hl7Ball]

Otro de los problemas que presentan los tipos de datos de HL7 tiene que ver con las colecciones. En HL7 estos tipos son *LIST*, *BAG*, *COLL*, *SET* y sus respectivos subtipos. Los atributos que pertenecen a alguno de ellos tienen una cardinalidad de 0..*, lo que en otras palabras quiere decir que representan una colección de colecciones, como por ejemplo el atributo *name* de la clase *Person* que se muestra en la figura 18.3.

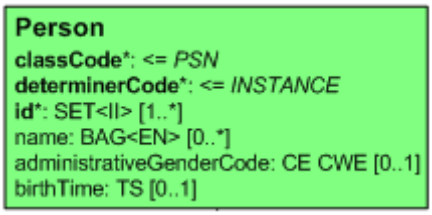


Figura 18.3 Clase Person de HL7, obtenida de [Hl7Ball]

El problema, es que si se miran los ficheros XSD de la implementación, todos estos tipos se convierten simplemente a colecciones, ya que por ejemplo un BAG<INT> con multiplicidad 0..* pasa a ser simplemente INT con 0..*, un SET<PQ> con multiplicidad 0..* pasa a ser un PQ con 0..*, etc. Esto nos indica que a la hora de definir las multiplicidades en los esquemas gráficos,

se cometió un error y todas las multiplicidades relativas a los conjuntos deberían ser en realidad 0..1 en lugar de 0..*.

Por este motivo, en los esquemas generados por la herramienta de conversión construida, todos los tipos que representan a colecciones pasan a ser tipos simples con multiplicidad 0..*. Así, el atributo *name* de la clase *Person* mostrada en la figura 18.3, pasaría a ser del tipo *EN* con una multiplicidad de 0..*.

Como se ha podido comprobar, la especificación de los tipos de datos contiene numerosos errores, y aquí sólo se han expuesto los más evidentes. Tal vez éste es uno de los motivos por los cuales hay tantas diferencias entre su especificación e implementación.

Corregir los errores comentados queda lejos del alcance de este proyecto, puesto que el esfuerzo necesario para solventar la problemática relacionada con los tipos de datos de HL7 sería suficiente como para constituir un proyecto por sí solo.

Teniendo en cuenta todos los inconvenientes descritos en esta sección, así como los tipos de datos ya expuestos en el capítulo 9, la parte del metamodelo de HL7 relativa a los tipos de datos se ha definido como se muestra en la figura 18.4. Hay que tener presente que la parte mostrada completa el metamodelo de HL7 mostrado en el capítulo 13, en el que respecto a los tipos de datos sólo se incluye la clase abstracta *DataType*.

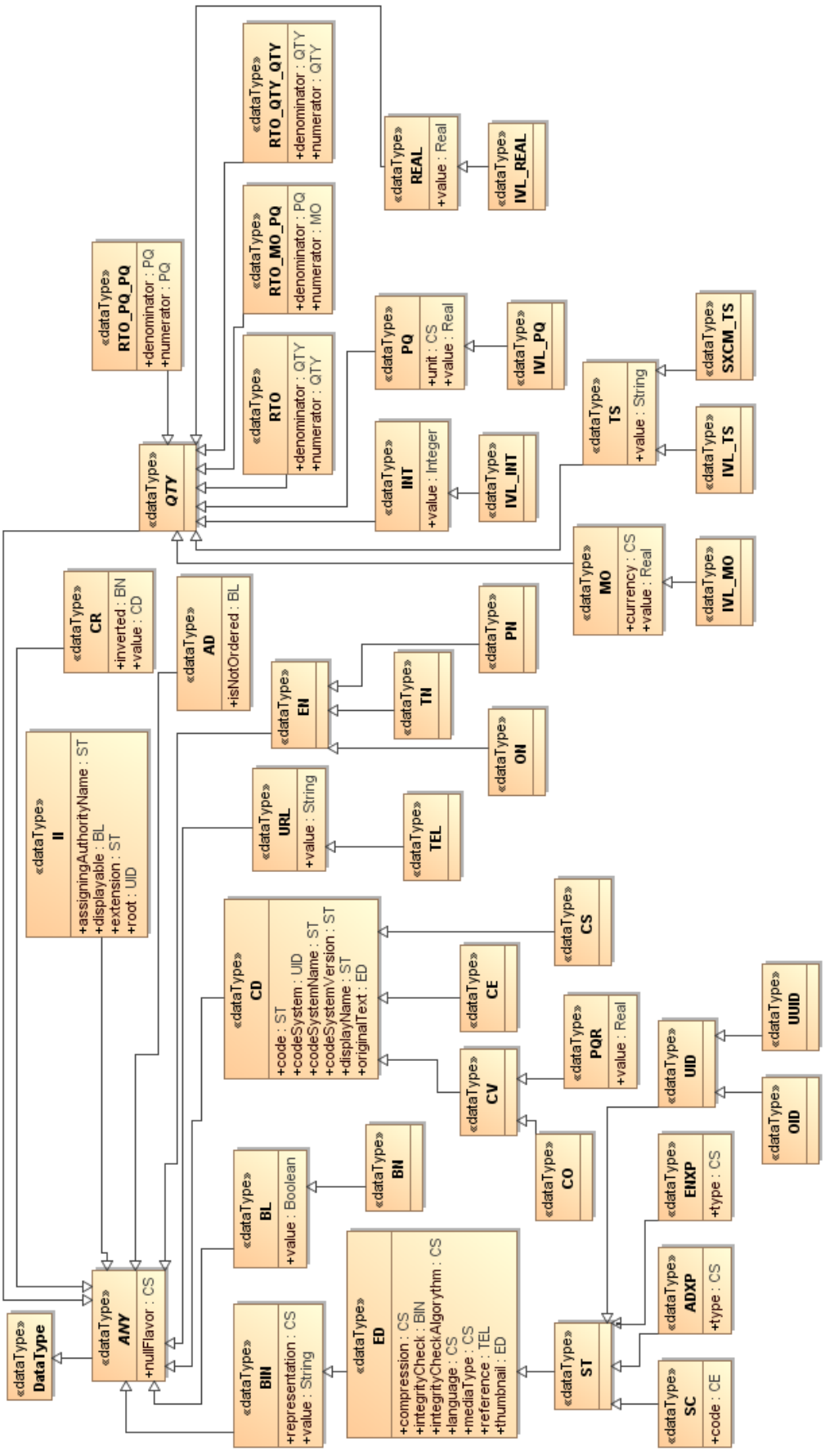


Figura 18.4 Fragmento del metamodelo de HL7 relativo a los tipos de datos

18.2 Restricciones textuales no procesables

Como ya se ha explicado en el bloque dedicado a HL7, algunos de los esquemas del estándar presentan restricciones textuales como la que se puede encontrar en la figura 18.5, que se refiere a la anulabilidad de alguno de los atributos de la clase *Person* extraída del esquema con identificador *COCT_RM030002UV*.

El problema es que en ninguno de los ficheros que contienen la información de esos esquemas en un formato procesable, se pueden encontrar dichas restricciones textuales.

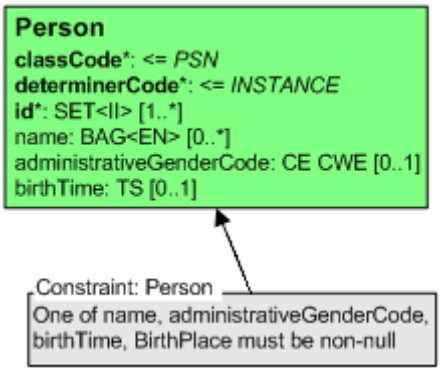


Figura 18.5 Restricción textual en HL7, obtenida de [Hl7Ball]

La decisión de no incluir dicha información, no es justificable de acuerdo a la tecnología disponible, puesto que incluirla en los ficheros MIF del estándar no debería resultar complicado, ya que al seguir la misma estructura que los ficheros XML bastaría con crear una nueva etiqueta que representase las restricciones textuales e incluir en ella su especificación.

Además, el hecho que las restricciones textuales no se incluyan en los ficheros MIF y no estén especificadas en un lenguaje procesable como podría ser OCL, supone irremediablemente que no se utilicen para nada en la práctica a la hora de intercambiar mensajes HL7 entre distintos sistemas de información. Esto es debido a que, obviamente, al estar sólo descritas en un esquema gráfico, no hay forma de que esas restricciones puedan validarse en las herramientas que tratan con el estándar HL7, a no ser que se opte por soluciones ad-hoc.

Este inconveniente hace que en nuestro proyecto, no podamos trabajar con las restricciones textuales, ya que resulta completamente inasumible tratarlas todas de forma manual, recordemos que existen unos 380 esquemas y que cada uno contiene unas dos o tres restricciones en promedio. Por este motivo, los esquemas UML resultantes de las conversiones no incluyen restricciones textuales.

No obstante, la herramienta desarrollada está diseñada de tal manera que, en caso que en un momento dado los comités de HL7 decidieran incluir en los ficheros MIF las restricciones, sea relativamente fácil incorporarlas a los esquemas UML que produce.

18.3 Nombres de clases repetidos

En algunos esquemas del estándar HL7, se pueden encontrar clases que representan conceptos diferentes, pero que poseen el mismo nombre. Así por ejemplo, en el R-MIM del estándar con identificador *COCT_RM030200UV*, se pueden encontrar la clase *Person* de la figura 18.6 y el CMET de la figura 18.7. El conflicto de nombres, viene dado porque la clase principal de ese CMET, mostrada en la figura 18.8, también se denomina *Person*, aunque ambas presenten atributos y asociaciones diferentes.

Es importante no confundir la situación aquí descrita, con otra expuesta en el apartado 8.2.8, en la que se utiliza una técnica consistente en representar en un mismo esquema la misma clase más de una vez para facilitar su lectura.

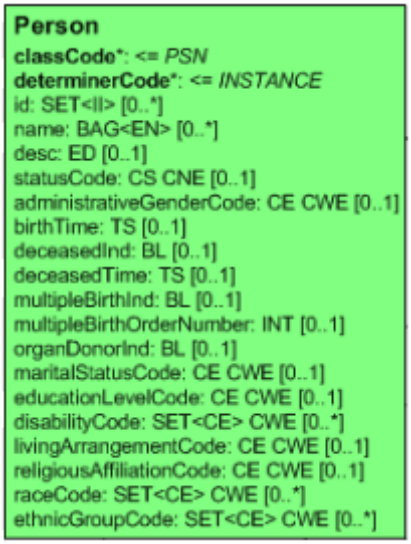


Figura 18.6 Clase Person del esquema COCT_RM030200UV, obtenida de [Hl7Ball]

Figura 18.7 CMET E_Person identified/confirmable, obtenido de [Hl7Ball]

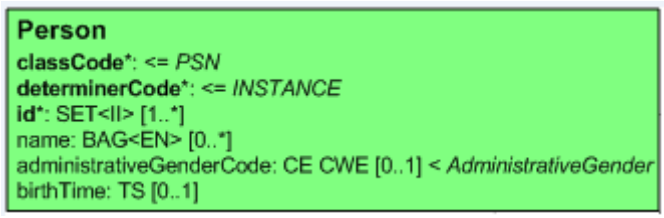


Figura 18.8 Clase Person del esquema COCT_RM030202UV, obtenida de [Hl7Ball]

Cabe destacar que la problemática presente en el ejemplo mostrado no es un hecho aislado, sino que existen múltiples esquemas del estándar en los que se repite esta misma situación.

Esta circunstancia, en nuestro caso, representa un problema, puesto que en UML no pueden coincidir dos clases diferentes con el mismo nombre en un único esquema conceptual.

La forma de tratar este problema en UML consistiría en definir una jerarquía de clases donde el elemento más general contuviese todos los atributos que comparten todas las clases que aparecen con el mismo nombre.

El problema que nos encontramos en este proyecto, es que siguiendo las normas de UML se deberían cambiar los nombres de muchas de las clases que aparecen en el estándar HL7, para que no se repitiesen nombres, y definir toda una serie de jerarquías o restricciones textuales que indicasen qué atributos deben presentar en cada caso.

Tratar este problema no es asumible dentro de nuestro proyecto, ya que volver a definir los nombres de la gran mayoría de las clases no es una tarea para nada trivial, y además se necesitaría de la ayuda de un experto en el ámbito de la salud para poder dotar a cada una de las clases de un nombre adecuado que identificase claramente el concepto representado.

Además, tal y como se ha afirmado anteriormente, nuestro objetivo es que los esquemas UML que produzca la herramienta que se ha desarrollado en este proyecto puedan ser utilizados por la gente que trabaja con el estándar HL7. Si se cambiasen los nombres de un gran número de clases, los esquemas UML resultantes no se parecerían en nada los del estándar HL7 y por tanto, no resultarían de utilidad alguna.

18.4 Formato de los ficheros MIF

Recordemos que el MIF es el formato de los ficheros de los cuales se ha extraído la información del estándar, basado en XML.

Uno de los inconvenientes que presenta este tipo de ficheros, es que según cuál se esté tratando, las etiquetas que hacen referencia a cada uno de los objetos se pueden encontrar representadas de distintas maneras.

Así por ejemplo, en algunos de ellos, la etiqueta que marca el inicio de la definición de una clase es <*class*>, mientras que en otros es <*mif:class*>. Este hecho, se repite prácticamente con cada una de las etiquetas, y en algunas además el nombre entero cambia por completo, en lugar de simplemente añadir el prefijo *mif*.

Este hecho, si bien no representa un gran inconveniente, sí que es cierto que incrementa la complejidad del código encargado de tratar los ficheros fuente del estándar, además que todos los esquemas no sigan un formato homogéneo resulta difícil de justificar.

18.5 Diferencias entre los ficheros procesables y los de formato gráfico

Otro de los puntos que se podría mejorar en el estándar, es el hecho que la información presentada por los ficheros que contienen la información de los esquemas conceptuales, como los MIF, en ocasiones no se corresponda completamente con la versión gráfica de los mismos.

Es cierto que en la mayoría de los casos esas diferencias no son muy acentuadas. El caso más común es encontrar que algunos nombres de clases presenten diferencias, pero también es cierto, que en ocasiones se pueden encontrar clases enteras que aparecen en la versión gráfica y no en la procesable, o viceversa.

De nuevo, este aspecto no dice mucho a favor del rigor que se le presupone a un estándar como HL7. Refleja que las versiones procesables y la gráfica, se generan por separado, cuando en un escenario normal se podría suponer que una se debería generar a partir de la otra con el objetivo de no replicar el trabajo y evitar la aparición de errores o diferencias entre ambas.

Es importante recalcar este aspecto, puesto que como consecuencia existirán esquemas conceptuales UML generados por la herramienta que se ha desarrollado en este proyecto que no coincidirán completamente con las versiones gráficas de los esquemas del estándar. Debido a que los esquemas UML siempre estarán basados en las versiones procesables, concretamente en los ficheros MIF anteriormente explicados.

19 EJEMPLO DE CONVERSIÓN

En este capítulo, se expone un ejemplo completo de conversión de esquema HL7 a UML. El motivo es que creemos que el hecho de disponer de un ejemplo en el que se expliquen los resultados obtenidos en cada una de las fases del proyecto de forma detallada, ayudará al lector a acabar de asimilar el funcionamiento de la herramienta construida.

El esquema conceptual del estándar HL7 escogido es *A_DrugIntervention* con identificador *PORR_RM049007UV*, que se muestra en la figura 19.1. En dicho esquema, se expresa la información relativa a un evento que involucra un suministro de medicamentos. Este ejemplo nos resultará útil puesto que, a diferencia de lo que ocurre en los que se han mostrado anteriormente, contiene algunas notas HL7 y además presenta algunos de los problemas que se han expuesto en el capítulo anterior.

En el esquema, la clase principal *SubstanceAdministrationEvent* representa el acto de haber suministrado un medicamento.

Dicha clase está relacionada con otras muchas de tipo *Participation*, que extienden la información proporcionada sobre ese acto. Así, *AuthorOrPerformer*, indica quién ha realizado la acción, *Location* dónde se ha realizado, *DirectTarget* cuál ha sido el medicamento suministrado y *Subject*, el sujeto de la acción, es decir, a quién se le ha suministrado.

La clase *SubstanceAdministrationEvent* también se relaciona con diversas de tipo *ActRelationship*. Existen dos reflexivas, *Component* y *PertinentInformation2*, que indican con qué otros actos de suministro del mismo medicamento está relacionada, en caso que se necesite administrar el mismo medicamento más de una vez al mismo sujeto. Por otro lado, *PertinentInformation1* sirve para indicar todos aquellos documentos clínicos que guardan algún tipo de relación con el acto de administración de medicamentos representado. Finalmente, *Support1* y *Support2* indican información relativa a la intervención y a las acciones realizadas durante el transcurso de dicho acto.

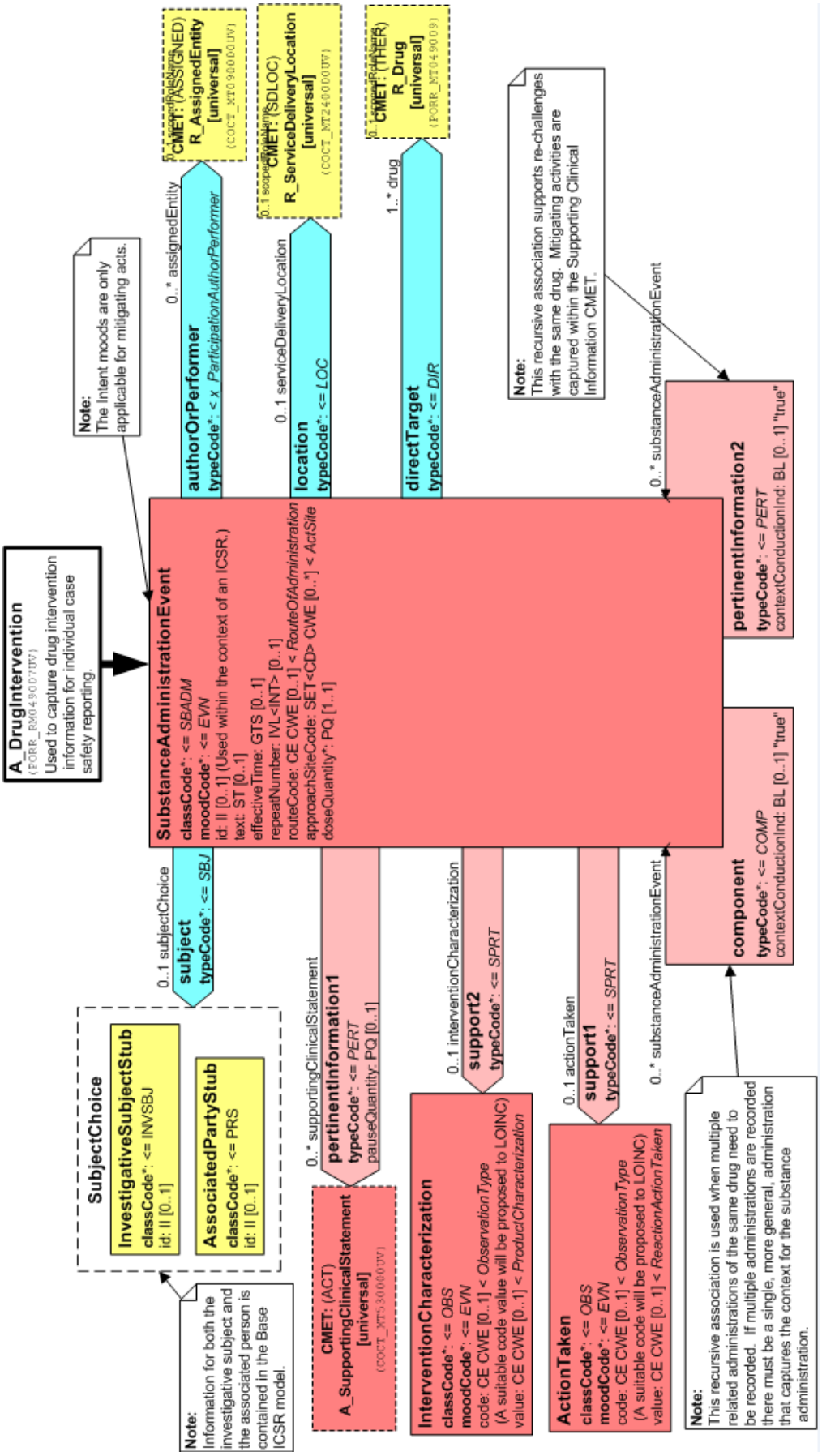


Figura 19.1 Esquema A_DrugIntervention, obtenido de [Hl7Ball]

El primer paso consiste en ejecutar el *parser* de ficheros MIF, para a partir del fichero fuente MIF que contiene la información del esquema conceptual mostrado, obtener un fichero XML instancia del metamodelo de HL7 diseñado anteriormente.

El resultado se muestra en el ejemplo 19.1. Para entender las explicaciones presentadas a continuación es necesario tener en cuenta la numeración de los identificadores ya explicada anteriormente. Recordemos que los identificadores se empiezan a asignar desde el primer elemento sin tener en cuenta las cabeceras (en el caso mostrado ese elemento es el 3) y que al primero se le asigna el identificador 0, al segundo el 1, y así consecutivamente.

A continuación, se exponen los detalles más relevantes del fichero XML:

Elementos del 3 al 16: definen las clases que aparecen en el esquema original. Vemos que cada una de ellas es un tipo diferente de elemento dependiendo de su tipo (*ActType*, *ParticipationType*, etc), y que contienen un atributo con su nombre (*name*) y otro con la lista de identificadores de los atributos que contiene (*ownedAttribute*).

Así por ejemplo, el elemento 9 representa la clase *Location*, que es de tipo *Participation* y contiene el atributo con identificador 65, que corresponde a su *typeCode*.

Elemento 17: contiene la información del único *choice* presente en el esquema, su nombre se define en el atributo *name*, que en este caso es SubjectChoice y los dos elementos que contiene en *ownedElement*, donde se encuentran los identificadores 2 y 3, que son los de los elementos que representan las clases *InvestigativeSubjectStub* y *AssociatedPartyStub* respectivamente.

Elementos del 18 al 21: se encuentran definidos los cuatro CMETs que aparecen en el esquema original. Para cada uno de ellos, se indica su nombre (*name*), el tipo de su clase principal (*mainClassType*) y los atributos *attributionLevel*, *rootClassCode* e *identifier*.

De este modo, en cuanto al CMET *R_AssignedEntity*, se puede comprobar que el tipo de su clase principal es *Role*, que coincide con lo expresado en el esquema original, ya que aparece pintado de amarillo. Para el resto de atributos es fácil comprobar que sus valores coinciden con los representados en el esquema fuente, se puede ver que su *rootClassCode* que en el esquema aparece representado entre paréntesis, toma por valor "*ASSIGNED",* su identificador es *COCT_MT090000UV01* y su *attributionLevel*, que aparece representado entre corchetes, "*universal*".

Elemento 22: contiene la información del *Entry Point*, concretamente su nombre, identificador, descripción y el elemento al que apunta, que en este caso es el que tiene como identificador 0 y que corresponde a la clase *SubstanceAdministrationEvent* de tipo *Act*.

Elementos 23, 26, 29, 31, 33, 35, 37 y 108: definen los tipos de datos que aparecen en el esquema mostrado. En concreto son: *CS*, *II*, *ST*, *SXCM_TS*, *IVL_INT*, *CE*, *CD*, *PQ* y *BL*.

Aquí se pueden apreciar algunas diferencias respecto a lo que contiene el esquema original, debido a que en los esquemas gráficos se utilizan los tipos de datos de la especificación UML del estándar, mientras que en los ficheros MIF, se utilizan los mismos que en el intercambio de mensajes. Este problema de discordancia ya fue comentado con detalle en el capítulo anterior.

En nuestro caso, las diferencias que provoca son las siguientes: el tipo de dato *IVL<INT>*, a nivel de implementación pasa a ser *IVL_INT*, *GTS* se convierte en *SXCM_TS* y *SET<CD>* lo expresamos como *CD* por la problemática descrita en el capítulo citado en el párrafo anterior.

Finalmente, hay que tener en cuenta que aunque en la versión gráfica no aparezca el tipo de dato *CS*, en el XML sí que se define, puesto que aunque no se presente de forma explícita, todos los atributos *typeCode* y *classCode* son de este tipo.

Elementos 27, 80 y 92: son las notas que representan los comentarios que contienen entre paréntesis los siguientes tres atributos: *id* de *SubstanceAdministrationEvent*, *code* de *InterventionCharacterization* y *code* de *ActionTaken*.

En cada uno de estos tres elementos, se encuentra un atributo *body*, que es el que contiene el texto de la nota y *type* que indica si dicha nota es de tipo descriptivo o de uso.

Para saber a qué elemento hace referencia cada una de las notas, hay que fijarse en el conjunto *note* definido en las clases y atributos. Así por ejemplo, la nota con identificador 27, que contiene el texto "*Used within the context of an ICSR.*", sabemos que hace referencia al atributo *id* de la clase *SubstanceAdministrationEvent*, porque en el elemento con identificador 28, que es el que representa a ese atributo, dentro de su conjunto denominado *note*, aparece el identificador 27.

Por otro lado, se puede comprobar que las notas que aparecen en el esquema original haciendo referencia al *choice SubjectChoice* y a las clases *Component*, *PertinentInformation2* y *SubstanceAdministrationEvent*, no aparecen en el fichero XML resultante, esto es debido al problema, también comentado en el capítulo anterior, que consiste en que en determinadas ocasiones no todos los elementos que se encuentran en la versión gráfica del esquema se

presentan también en el fichero MIF o viceversa. En el caso del esquema que nos ocupa, esas cuatro notas aparecen en la versión gráfica del esquema, pero no en el fichero MIF que tiene asociado.

Resto de elementos: son de tipo *Property* o *Association*. Recordemos que las *properties* pueden jugar dos roles: atributo de una clase y extremo de una asociación. Las primeras, son aquellas que se referencian desde el conjunto *ownedAttribute* de las clases definidas al principio del fichero XML.

Así por ejemplo, vemos que los dos atributos de la clase *AssociatedPartyStub* de tipo *Role*, aparecen representados por los elementos con identificador 45 y 46, que son dos *properties* con nombre *classCode* e *Id* respectivamente.

Por otro lado, aquellas que actúan como extremos de una asociación, aparecen referenciadas desde el conjunto con nombre *memberEnd* que poseen los elementos de tipo *Association*. Por ejemplo, la asociación existente entre las clases *SubstanceAdministrationEvent* y *Location*, aparece representada por el elemento 66 y sus dos extremos por los elementos 67 y 72.

Ambos tipos de *properties* presentan una multiplicidad y un tipo. En cuanto a la multiplicidad, cabe destacar que si el límite inferior es 0 no aparece representado, y en cuanto al tipo, que el de las que actúan como extremos de una asociación es una clase, *choice* o CMET, mientras que el de las que juegan el papel de atributos de una clase, será siempre un tipo de dato.

```
1. <?xml version="1.0" encoding="ASCII"?>
2. <xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:hl7_metamodel="http://hl7_metamodel/1.0">
3.   <hl7_metamodel:ActType name="SubstanceAdministrationEvent"
      ownedAttribute="/21 /22 /25 /27 /29 /31 /33 /35 /37"/>
4.   <hl7_metamodel:ParticipationType name="Subject"
      ownedAttribute="/40"/>
5.   <hl7_metamodel:RoleType name="InvestigativeSubjectStub"
      ownedAttribute="/43 /44"/>
6.   <hl7 metamodel:RoleType name="AssociatedPartyStub"
      ownedAttribute="/45 /46"/>
7.   <hl7_metamodel:ParticipationType name="DirectTarget"
      ownedAttribute="/51"/>
8.   <hl7_metamodel:ParticipationType name="AuthorOrPerformer"
      ownedAttribute="/58"/>
9.   <hl7_metamodel:ParticipationType name="Location"
      ownedAttribute="/65"/>
10.  <hl7_metamodel:ActRelationshipType name="Support1"
      ownedAttribute="/72"/>
11.  <hl7_metamodel:ActType name="ActionTaken" ownedAttribute="/75
      /76 /78 /79"/>
12.  <hl7_metamodel:ActRelationshipType name="Support2"
      ownedAttribute="/84"/>
13.  <hl7_metamodel:ActType name="InterventionCharacterization"
```

```
      ownedAttribute="/87 /88 /90 /91"/>
14.   <hl7_metamodel:ActRelationshipType name="PertinentInformation1"
      ownedAttribute="/96 /97"/>
15.   <hl7_metamodel:ActRelationshipType name="PertinentInformation2"
      ownedAttribute="/104 /106"/>
16.   <hl7_metamodel:ActRelationshipType name="Component"
      ownedAttribute="/113 /114"/>
17.   <hl7_metamodel:Choice name="SubjectChoice" ownedElement="/2
      /3"/>
18.   <hl7_metamodel:CMET name="R_DrugUniversal" rootClassCode="ROL"
      attributionLevel="universal" identifier="PORR_MT049009UV01"
      mainClassType="Role"/>
19.   <hl7_metamodel:CMET name="R_AssignedEntityUniversal"
      rootClassCode="ASSIGNED" attributionLevel="universal"
      identifier="COCT_MT090000UV01" mainClassType="Role"/>
20.   <hl7_metamodel:CMET name="R_ServiceDeliveryLocationUniversal"
      rootClassCode="SDLOC" attributionLevel="universal"
      identifier="COCT_MT240000UV01" mainClassType="Role"/>
21.   <hl7_metamodel:CMET name=
      "A_SupportingClinicalStatementUniversal" rootClassCode="ACT"
      attributionLevel="universal" identifier="COCT_MT530000UV"
      mainClassType="Act"/>
22.   <hl7_metamodel:EntryPoint name="A_DrugInterventionUniversal"
      identifier="PORR_MT049007UV01" description="Used to capture
      drug intervention information for individual case safety
      reporting." choosableElement="/0"/>
23.   <hl7_metamodel:CS name="CS"/>
24.   <hl7_metamodel:Property name="classCode" type="/20" upper="1"
      lower="1" codingStrength="CNE" mnemonic="SBADM"
      codeSystemName="ActClass"/>
25.   <hl7_metamodel:Property name="moodCode" type="/20" upper="1"
      lower="1" codingStrength="CNE" mnemonic="EVN"
      codeSystemName="ActMood"/>
26.   <hl7_metamodel:II name="II"/>
27.   <hl7_metamodel:Note body="Used within the context of an ICSR."
      type="Usage"/>
28.   <hl7_metamodel:Property note="/24" name="id" type="/23"
      upper="1"/>
29.   <hl7_metamodel:ST name="ST"/>
30.   <hl7_metamodel:Property name="text" type="/26" upper="1"/>
31.   <hl7_metamodel:SXCM_TS name="SXCM_TS"/>
32.   <hl7_metamodel:Property name="effectiveTime" type="/28"
      upper="1"/>
33.   <hl7_metamodel:IVL_INT name="IVL_INT"/>
34.   <hl7_metamodel:Property name="repeatNumber" type="/30"
      upper="1"/>
35.   <hl7_metamodel:CE name="CE"/>
36.   <hl7_metamodel:Property name="routeCode" type="/32" upper="1"
      codingStrength="CWE" domainName="RouteOfAdministration"/>
37.   <hl7_metamodel:CD name="CD"/>
38.   <hl7_metamodel:Property name="approachSiteCode" type="/34"
      upper="-1" codingStrength="CWE" domainName="ActSite"/>
39.   <hl7_metamodel:PQ name="PQ"/>
40.   <hl7_metamodel:Property name="doseQuantity" type="/36" upper="1"
      lower="1"/>
41.   <hl7_metamodel:Association memberEnd="/48 /39"/>
42.   <hl7_metamodel:Property name="subject" type="/1" upper="1"/>
43.   <hl7_metamodel:Property name="typeCode" type="/20" upper="1"
      lower="1" codingStrength="CNE" mnemonic="SBJ"
      codeSystemName="ParticipationType"/>
44.   <hl7_metamodel:Association memberEnd="/47 /42"/>
```

```
45.  <hl7_metamodel:Property name="subjectChoice" type="/14"
      upper="1" lower="1"/>
46.  <hl7_metamodel:Property name="classCode" type="/20" upper="1"
      lower="1" codingStrength="CNE" mnemonic="INVSBJ"
      default="INVSBJ" codeSystemName="RoleClass"/>
47.  <hl7_metamodel:Property name="id" type="/23" upper="1"/>
48.  <hl7_metamodel:Property name="classCode" type="/20" upper="1"
      lower="1" codingStrength="CNE" mnemonic="PRS" default="PRS"
      codeSystemName="RoleClass"/>
49.  <hl7_metamodel:Property name="id" type="/23" upper="1"/>
50.  <hl7_metamodel:Property name="participation" type="/1"
      upper="-1"/>
51.  <hl7_metamodel:Property name="act" type="/0" upper="1"
      lower="1"/>
52.  <hl7_metamodel:Association memberEnd="/55 /50"/>
53.  <hl7_metamodel:Property name="directTarget" type="/4"
      upper="-1" lower="1"/>
54.  <hl7_metamodel:Property name="typeCode" type="/20" upper="1"
      lower="1" codingStrength="CNE" mnemonic="DIR"
      codeSystemName="ParticipationType"/>
55.  <hl7_metamodel:Association memberEnd="/54 /53"/>
56.  <hl7_metamodel:Property name="manufacturedProduct2" type="/15"
      upper="1" lower="1"/>
57.  <hl7_metamodel:Property name="participation" type="/4"
      upper="-1"/>
58.  <hl7_metamodel:Property name="act" type="/0" upper="1"
      lower="1"/>
59.  <hl7_metamodel:Association memberEnd="/62 /57"/>
60.  <hl7_metamodel:Property name="authorOrPerformer" type="/5"
      upper="-1"/>
61.  <hl7_metamodel:Property name="typeCode" type="/20" upper="1"
      lower="1" codingStrength="CNE"
      domainName="x_ParticipationAuthorPerformer"/>
62.  <hl7_metamodel:Association memberEnd="/61 /60"/>
63.  <hl7_metamodel:Property name="assignedEntity" type="/16"
      upper="1" lower="1"/>
64.  <hl7_metamodel:Property name="participation" type="/5"
      upper="-1"/>
65.  <hl7_metamodel:Property name="act" type="/0" upper="1"
      lower="1"/>
66.  <hl7_metamodel:Association memberEnd="/69 /64"/>
67.  <hl7_metamodel:Property name="location" type="/6" upper="1"/>
68.  <hl7_metamodel:Property name="typeCode" type="/20" upper="1"
      lower="1" codingStrength="CNE" mnemonic="LOC" default="LOC"
      codeSystemName="ParticipationType"/>
69.  <hl7_metamodel:Association memberEnd="/68 /67"/>
70.  <hl7_metamodel:Property name="serviceDeliveryLocation"
      type="/17" upper="1" lower="1"/>
71.  <hl7_metamodel:Property name="participation" type="/6"
      upper="-1"/>
72.  <hl7_metamodel:Property name="act" type="/0" upper="1"
      lower="1"/>
73.  <hl7_metamodel:Association memberEnd="/81 /71"/>
74.  <hl7_metamodel:Property name="support1" type="/7" upper="1"/>
75.  <hl7_metamodel:Property name="typeCode" type="/20" upper="1"
      lower="1" codingStrength="CNE" mnemonic="SPRT" default="SPRT"
      codeSystemName="ActRelationshipType"/>
76.  <hl7_metamodel:Association memberEnd="/80 /74"/>
77.  <hl7_metamodel:Property name="actionTaken" type="/8" upper="1"
      lower="1"/>
78.  <hl7_metamodel:Property name="classCode" type="/20" upper="1"
```

```
      lower="1" codingStrength="CNE" mnemonic="OBS"
      codeSystemName="ActClass"/>
79.   <hl7_metamodel:Property name="moodCode" type="/20" upper="1"
      lower="1" codingStrength="CNE" mnemonic="EVN"
      codeSystemName="ActMood"/>
80.   <hl7_metamodel:Note body="A suitable code will be proposed to
      LOINC" type="Usage"/>
81.   <hl7_metamodel:Property note="/77" name="code" type="/32"
      upper="1" codingStrength="CWE" domainName="ObservationType"/>
82.   <hl7_metamodel:Property name="value" type="/32" upper="1"
      codingStrength="CWE" domainName="ReactionActionTaken"/>
83.   <hl7_metamodel:Property name="inboundRelationship" type="/7"
      upper="-1"/>
84.   <hl7_metamodel:Property name="source" type="/0" upper="1"
      lower="1"/>
85.   <hl7_metamodel:Association memberEnd="/93 /83"/>
86.   <hl7_metamodel:Property name="support2" type="/9" upper="1"/>
87.   <hl7_metamodel:Property name="typeCode" type="/20" upper="1"
      lower="1" codingStrength="CNE" mnemonic="SPRT" default="SPRT"
      codeSystemName="ActRelationshipType"/>
88.   <hl7_metamodel:Association memberEnd="/92 /86"/>
89.   <hl7_metamodel:Property name="interventionCharacterization"
      type="/10" upper="1" lower="1"/>
90.   <hl7_metamodel:Property name="classCode" type="/20" upper="1"
      lower="1" codingStrength="CNE" mnemonic="OBS"
      codeSystemName="ActClass"/>
91.   <hl7_metamodel:Property name="moodCode" type="/20" upper="1"
      lower="1" codingStrength="CNE" mnemonic="EVN"
      codeSystemName="ActMood"/>
92.   <hl7_metamodel:Note body="A suitable code value will be
      proposed to LOINC" type="Usage"/>
93.   <hl7_metamodel:Property note="/89" name="code" type="/32"
      upper="1" codingStrength="CWE" domainName="ObservationType"/>
94.   <hl7_metamodel:Property name="value" type="/32" upper="1"
      codingStrength="CWE" domainName="ProductCharacterization"/>
95.   <hl7_metamodel:Property name="inboundRelationship" type="/9"
      upper="-1"/>
96.   <hl7_metamodel:Property name="source" type="/0" upper="1"
      lower="1"/>
97.   <hl7_metamodel:Association memberEnd="/101 /95"/>
98.   <hl7_metamodel:Property name="pertinentInformation1" type="/11"
      upper="-1"/>
99.   <hl7_metamodel:Property name="typeCode" type="/20" upper="1"
      lower="1" codingStrength="CNE" mnemonic="PERT"
      codeSystemName="ActRelationshipType"/>
100.  <hl7_metamodel:Property name="pauseQuantity" type="/36"
      upper="1"/>
101.  <hl7_metamodel:Association memberEnd="/100 /99"/>
102.  <hl7_metamodel:Property name="clinicalStatement" type="/18"
      upper="1" lower="1"/>
103.  <hl7_metamodel:Property name="inboundRelationship" type="/11"
      upper="-1"/>
104.  <hl7_metamodel:Property name="source" type="/0" upper="1"
      lower="1"/>
105.  <hl7_metamodel:Association memberEnd="/110 /103"/>
106.  <hl7_metamodel:Property name="pertinentInformation2" type="/12"
      upper="-1"/>
107.  <hl7_metamodel:Property name="typeCode" type="/20" upper="1"
      lower="1" codingStrength="CNE" mnemonic="PERT"
      codeSystemName="ActRelationshipType"/>
108.  <hl7_metamodel:BL name="BL"/>
```

```
109. <hl7_metamodel:Property name="contextConductionInd" type="/105"
      upper="1" default="true"/>
110. <hl7_metamodel:Association memberEnd="/109 /108"/>
111. <hl7_metamodel:Property name="substanceAdministrationEvent"
      type="/0" upper="1" lower="1"/>
112. <hl7_metamodel:Property name="inboundRelationship" type="/12"
      upper="-1"/>
113. <hl7_metamodel:Property name="source" type="/0" upper="1"
      lower="1"/>
114. <hl7_metamodel:Association memberEnd="/118 /112"/>
115. <hl7_metamodel:Property name="component" type="/13" upper="-1"/>
116. <hl7_metamodel:Property name="typeCode" type="/20" upper="1"
      lower="1" codingStrength="CNE" mnemonic="COMP"
      codeSystemName="ActRelationshipType"/>
117. <hl7_metamodel:Property name="contextConductionInd" type="/105"
      upper="1" default="true"/>
118. <hl7_metamodel:Association memberEnd="/117 /116"/>
119. <hl7_metamodel:Property name="substanceAdministrationEvent"
      type="/0" upper="1" lower="1"/>
120. <hl7_metamodel:Property name="inboundRelationship" type="/13"
      upper="-1"/>
121. <hl7_metamodel:Property name="source" type="/0" upper="1"
      lower="1"/>
122.</xmi:XMI>
```

Ejemplo 19.1 Fichero XML del esquema A_DrugIntervention

Una vez se ha obtenido el fichero XML, el siguiente paso consiste en ejecutar las transformaciones ATL, cuya salida es un esquema UML.

Dicho esquema UML, se puede encontrar en la figura 19.3, tal y como se muestra en la herramienta *MagicDraw*, concretamente en su versión 16.8. A continuación, se exponen sus características más destacables:

El *Entry Point* del esquema fuente, en el destino aparece representado mediante una clase abstracta (*A_DrugInterventionUniversal*) a la que se le ha aplicado el estereotipo *Entry Point* y que contiene un atributo *description*, que toma por valor la misma descripción presente en el esquema original.

Por lo que respecta a las clases, cada una de las definidas en el esquema original se ha convertido en otra con el mismo nombre en el esquema destino. Además, cada una de ellas aparece estereotipada según su tipo. De este modo, a la clase *SubstanceAdministrationEvent* se le ha aplicado el estereotipo *Act*, y a *Location*, el estereotipo *Participation*.

En cuanto a los atributos, al igual que ocurre con las clases, por cada uno de los existentes en el esquema fuente, se ha creado uno en el destino. Se puede comprobar además, que aquellos que identifican la clase que los contiene, es decir, *classCode* y *typeCode*, aparecen definidos como de sólo lectura. Por otro lado, cabe destacar que aquellos que son de tipo código, como

por ejemplo *routeCode* de la clase *SubstanceAdministrationEvent*, presentan entre los caracteres '{' y '}' toda la información relativa a ese código, esto es: su *codingStrength*, *mnemonic*, *codeSystemName* y valor por defecto, siempre y cuando no sean nulos.

En lo referente a las asociaciones, por cada una de las definidas en el esquema de origen, también se crea una en el destino. De este modo, en el esquema UML, se puede observar que existen asociaciones, por ejemplo, entre las clases *SubstanceAdministrationEvent* y *Location* y entre *SubstanceAdministrationEvent* y *DirectTarget*.

En cuanto al *choice SubjectChoice* presente en el esquema original, se ha convertido en una clase abstracta con el estereotipo *Choice* aplicado y con el mismo nombre que tenía. Además, se ha definido una jerarquía donde dicha clase es la más general, y las más específicas son las dos que representan a los elementos contenidos en el *choice*, es decir, *InvestigativeSubjectStub* y *AssociatedPartyStub*.

Por lo que respecta a los cuatro CMETs que se pueden encontrar en el esquema HL7, se han generado cuatro clases a las que se les han aplicado dos estereotipos, uno es *cmet*, y el otro varía según el tipo de su elemento principal (*Act*, *Role*, etc). De este modo, a la clase R_DrugUniversal, se le han aplicado los estereotipos *cmet* y *Role*, vemos que esto coincide con lo definido en el esquema HL7, puesto que dicho CMET aparece de color amarillo, el correspondiente a la clase Role. Por último, cabe destacar que las clases que representan a los CMET contienen un atributo de sólo lectura que toma por valor el identificador del esquema conceptual de HL7 en el que se halla definida toda la información de dicho CMET.

Por último, cabe destacar que las notas HL7 se han convertido en comentarios UML. En la imagen no se pueden apreciar, puesto que la herramienta MagicDraw los oculta. Para poder demostrar que realmente se generan, se adjunta una captura de la vista jerárquica proporcionada por *Eclipse* en la figura 19.2, donde se puede apreciar el que se refiere al atributo *Id* de la clase *SubstanceAdministrationEvent*.

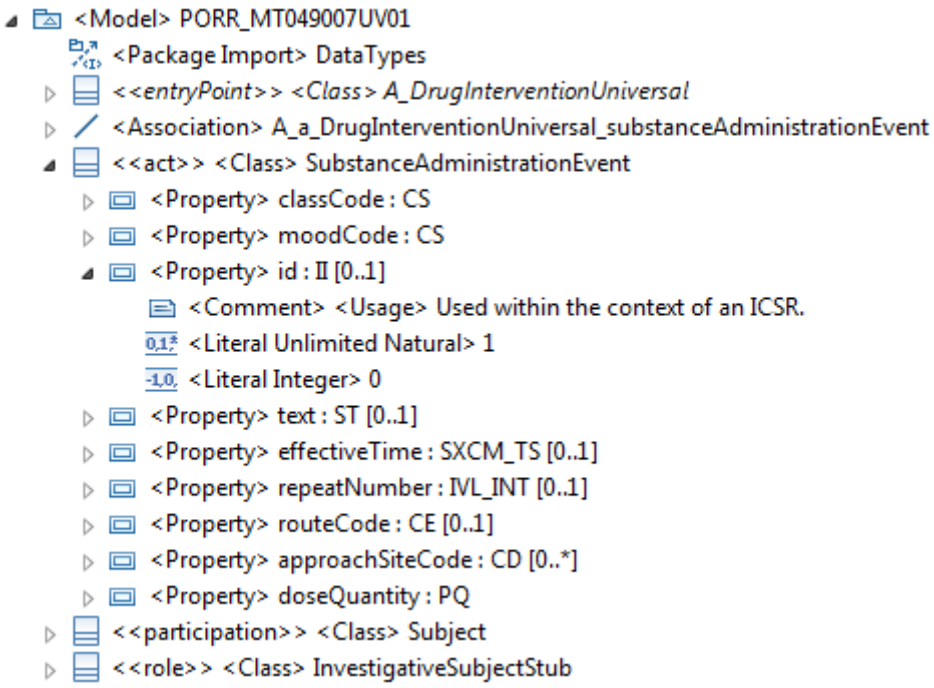


Figura 19.2 Vista jerárquica de Eclipse donde se puede ver un Comment de UML

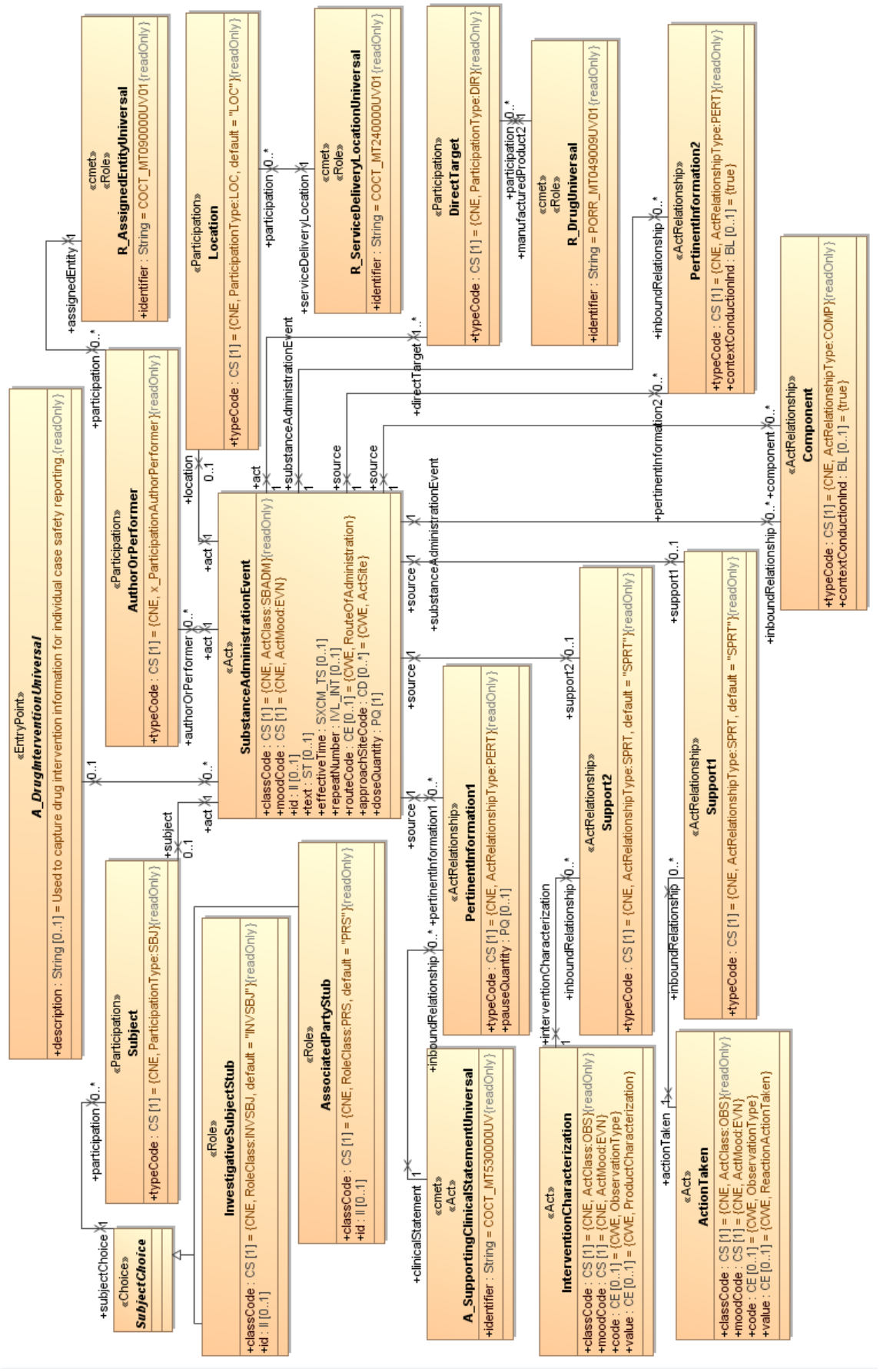


Figura 19.3 Esquema A_DrugIntervention convertido a UML

Una vez se ha obtenido el esquema conceptual UML, lo único que queda por hacer es ejecutar la herramienta que produce la versión refinada de los esquemas que se obtienen a partir de la ejecución de las reglas ATL.

El resultado obtenido se muestra en la figura 19.4. Lo único que diferencia a este esquema del anterior, es que los CMET se han sustituido por sus elementos principales. De este modo, los cambios realizados son cuatro:

- Se ha sustituido *A_SupportingClinicalStatement* por el *choice ClinicalStatement* del esquema con identificador *COCT_MT530000UV*.
- Se ha sustituido *R_AssignedEntityUniversal* por la clase *AssignedEntity* de tipo *Role* del esquema con identificador *COCT_MT090000UV01*.
- Se ha sustituido *R_ServiceDeliveryLocationUniversal* por la clase *ServiceDeliveryLocation* de tipo *Role* del esquema con identificador *COCT_MT240000UV01*.
- Se ha sustituido *R_DrugUniversal* por la clase *ManufacturedProduct2* de tipo *Role* del esquema con identificador *PORR_MT049009UV01*.

Además, aunque no se puede apreciar en la versión gráfica del esquema, se han corregido los enlaces que ATL había definido utilizando rutas que sólo eran válidas dentro de la plataforma *Eclipse*.

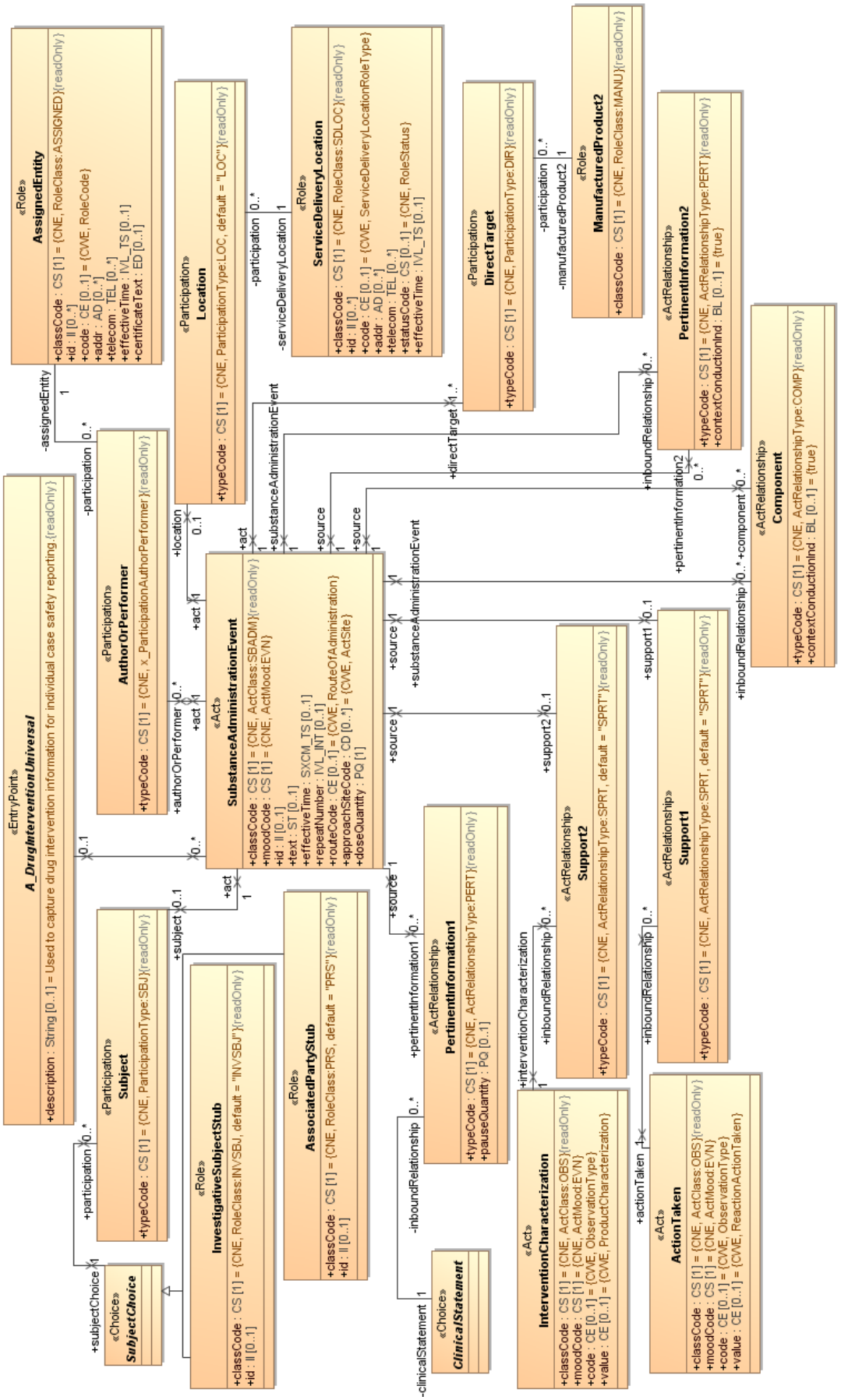


Figura 19.4 Esquema A_DrugIntervention convertido a UML y refinado

20 GUÍA DEL USUARIO

En este capítulo, se explica cómo ejecutar la herramienta de transformación de modelos HL7 a UML desarrollada en este proyecto.

El único requisito necesario para poder seguir las instrucciones proporcionadas, es tener instalado *Eclipse*. Se recomienda utilizar la versión 3.6, que es la que se ha empleado en este proyecto, para asegurar que no haya problemas de compatibilidad entre las versiones utilizadas. Además, se necesita el paquete que incluye los plugins relativos al modelado, que se puede encontrar en [EclMod] y que incluye todo lo relacionado con ATL.

20.1 Parser de ficheros MIF del estándar

Lo primero que se debe hacer, es ejecutar el proyecto que transforma los ficheros MIF del estándar en ficheros XML que son instancias del metamodelo de HL7 diseñado. Los ficheros fuente del proyecto Java que implementa esta fase, se encuentran en el directorio *HL7ToUML/mifparser_genmodel/src* del fichero zip que acompaña a este documento.

Para cargar el proyecto en *Eclipse*, hay que seleccionar la opción *Import* del menú *File*, seguidamente seleccionar la opción *Existing Projects into Workspace* 20.1 y para terminar, seleccionar el fichero *HL7ToUML/mifparser_genmodel.zip* tal y como se indica en el cuadro de texto de la figura 20.2.

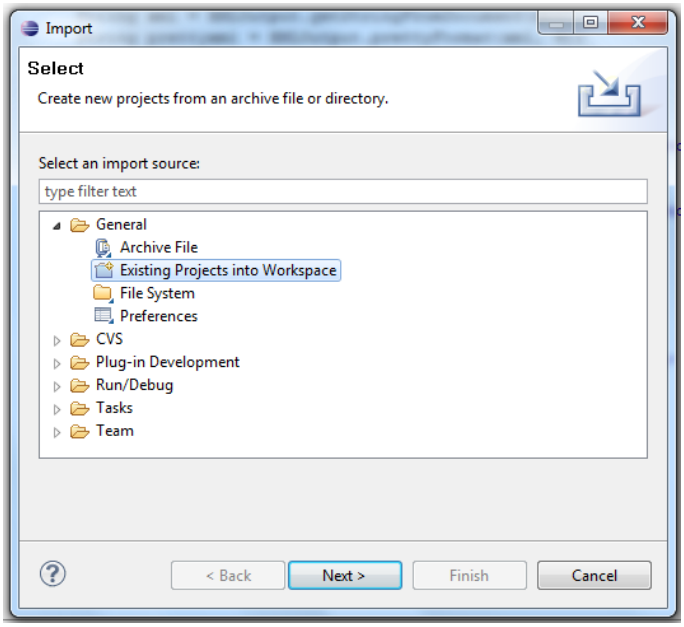


Figura 20.1 Menú Import de Eclipse

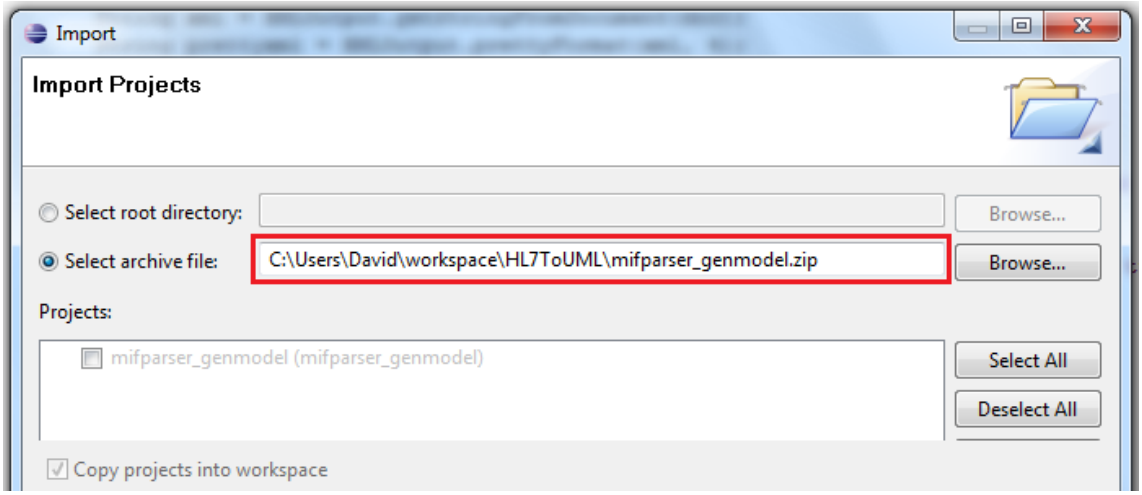


Figura 20.2 Selección del proyecto a importar en Eclipse

Todos los ficheros MIF del estándar que se deseen convertir, deben encontrarse en el directorio *HL7ToUML/mifparser_genmodel/mif*. Por otro lado, los XML, se generarán dentro de *HL7ToUML/mifparser_genmodel/hl7*.

Para realizar la transformación, se debe seleccionar la opción *run configurations* dentro del menú *run* de Eclipse. En la nueva ventana, hay que seleccionar el archivo Java del proyecto que nos ocupa, *MifParser*, y seguidamente en la pestaña *arguments*, introducir el nombre de todos los ficheros MIF que se desean convertir (figura 20.3).

Para facilitar esta tarea, se ofrece una lista con todos los ficheros MIF incluidos en la versión normativa del año 2009 en el fichero *HL7ToUML/mifparser_genmodel/@listaMIF.txt*.

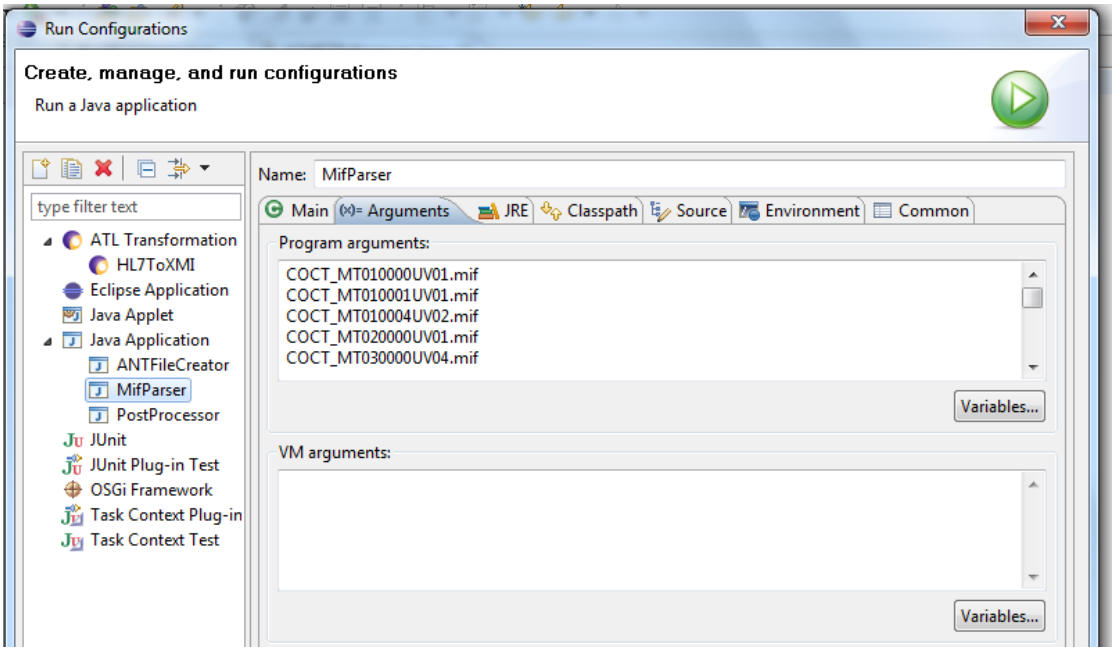


Figura 20.3 Configuración de ejecución del parser MIF

20.2 Tranformaciones ATL

El siguiente paso, tras haber obtenido los ficheros XML, consiste en ejecutar las transformaciones ATL. Todo lo relativo a esta fase, se encuentra en el directorio *HL7ToUML/HL7ToXMI* y el proyecto se carga de la misma forma que el anterior.

Los ficheros XML obtenidos en el paso anterior, deben copiarse en el directorio *HL7ToUML/HL7ToXMI /hl7*, mientras que los ficheros resultantes de la ejecución, se generarán en *HL7ToUML/HL7ToXMI /resultados*.

Para ejecutar las transformaciones que se han definido en ATL, se debe seleccionar la opción *run configurations* dentro del menú *run* de Eclipse. En la figura 20.4, se puede apreciar cómo debe configurarse la ejecución.

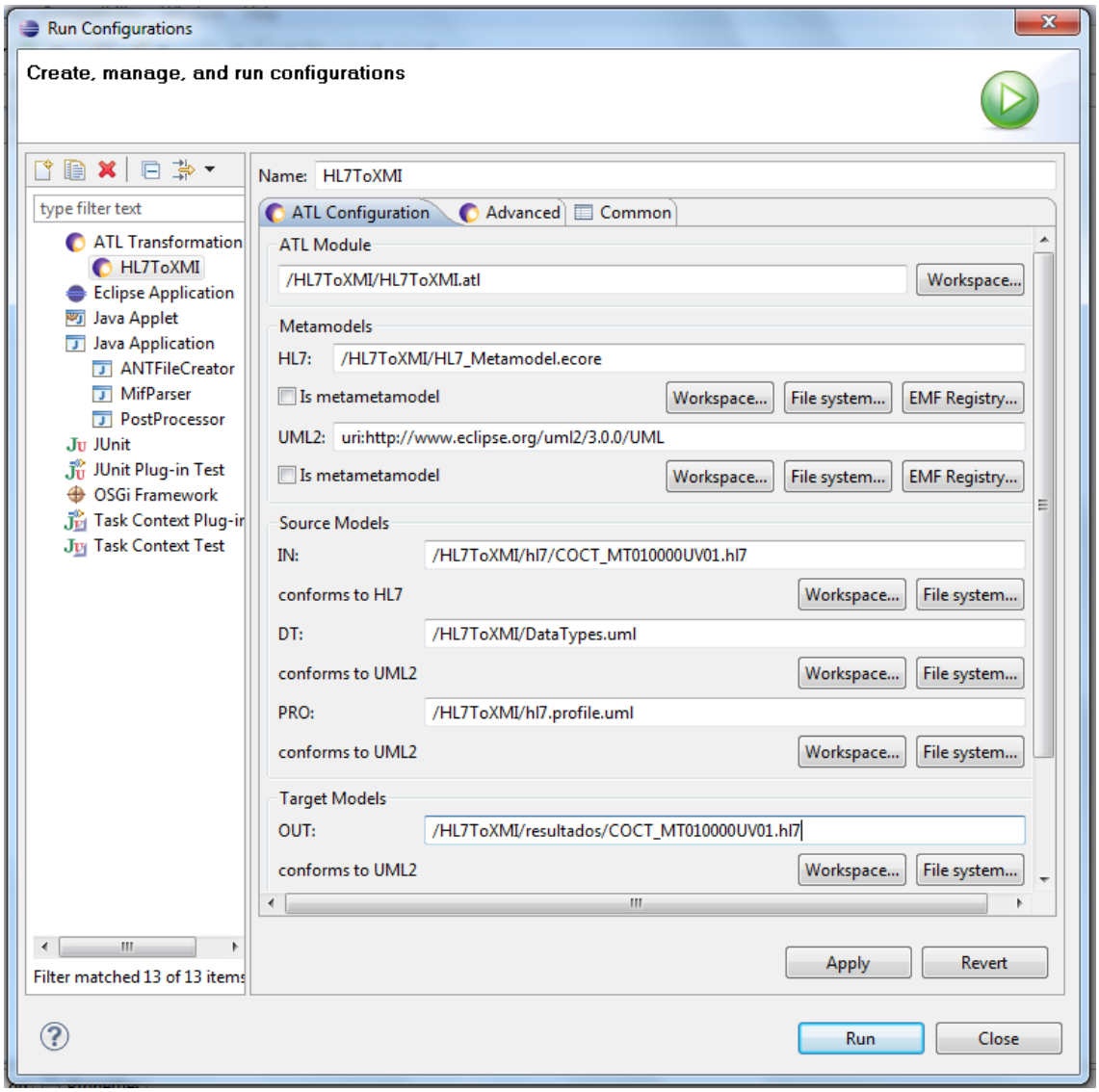


Figura 20.4 Configuración de ejecución de las reglas ATL

En el apartado *models*, en la caja de texto HL7, se debe indicar la ruta del metamodelo de HL7, mientras que en la de UML2, debe indicarse la del metamodelo de UML. Para esto último, hay que pulsar el botón *EMF Registry* y seleccionar el que se puede ver en la figura donde se muestra la configuración.

Por otro lado, dentro de la sección de *source models*, en la caja de texto IN, debe indicarse la versión XML del modelo de HL7 que se desea convertir, en DT el esquema UML donde se encuentran definidos los tipos de datos, y finalmente, en PRO el que contiene el *profile*.

Para terminar, en el apartado *target models*, debe indicarse la ruta del esquema UML resultante de la transformación.

Como se puede observar, siguiendo este método, las transformaciones deben ir ejecutándose una a una, y teniendo en cuenta que el número de esquemas que deseamos tratar ronda los 380, está claro que se debe buscar un método alternativo que permita tratarlos todos a la vez.

Afortunadamente, ATL nos ofrece la posibilidad de definir un archivo XML que contenga la información necesaria para poder realizar múltiples transformaciones en una sola ejecución.

Este fichero XML se encuentra en *HL7ToUML/HL7ToXMI /build.xml*. Su estructura es simple, primero se definen las cabeceras del fichero, tal y como se puede apreciar en el ejemplo 20.1.

```
<?xml version="1.0"?>
<project name="HL7ToXMI" default="run" basedir=".">
  <target name="run">
```

Ejemplo 20.1 Cabecera del fichero build.xml

Seguidamente, se definen todos los modelos que deben cargarse para realizar las transformaciones. Esto se realiza tal y como se indica en el ejemplo 20.2.

```
<atl.loadModel name="UML2" metamodel="%UML2"
 nsUri="http://www.eclipse.org/uml2/3.0.0/UML" />
<atl.loadModel name="HL7" metamodel="%HL7"
 nsUri="platform:/resource/HL7ToXMI/HL7_Metamodel.ecore" />
<atl.loadModel name="PRO" metamodel="%UML2"
 nsUri="platform:/resource/HL7ToXMI/hl7.profile.uml" />
<atl.loadModel name="DT" metamodel="%UML2"
 nsUri="platform:/resource/HL7ToXMI/DataTypes.uml" />
<atl.loadModel name="IN1" metamodel="%HL7"
 nsUri="platform:/resource/HL7ToXMI/hl7/COCT_MT010000UV01.hl7" />
<atl.loadModel name="IN2" metamodel="%HL7"
 nsUri="platform:/resource/HL7ToXMI/hl7/COCT_MT010001UV01.hl7" />
<atl.loadModel name="IN3" metamodel="%HL7"
 nsUri="platform:/resource/HL7ToXMI/hl7/COCT_MT010004UV02.hl7" />
```

Ejemplo 20.2 Definición de los modelos a cargar en el fichero build.xml

Como se puede comprobar, primero se definen los metamodelos UML2 y HL7. Seguidamente, los esquemas que contienen la información del *profile* y de los tipos de datos, que son PRO y DT respectivamente. Finalmente, se indican todos los esquemas HL7 en formato XML que se desean convertir. Como son demasiados, en el fragmento mostrado sólo se muestran los tres primeros.

Una vez se han indicado los esquemas, deben definirse las transformaciones que se desean realizar a partir de ellos. En el ejemplo 20.3, se puede observar cómo se define el caso de convertir el esquema con identificador *COCT_MT010001UV01*, que en el fragmento XML mostrado anteriormente, se identifica mediante el nombre *IN1*.

En la etiqueta *option*, se debe activar la opción de permitir referencias entre distintos modelos, puesto que recordemos, que en el esquema UML resultante, se referencia a los que contienen la información de los tipos de datos y el *profile*.

Por otro lado, en las etiquetas *inmodel*, se definen los modelos de entrada ya comentados anteriormente. En este caso, el modelo IN es el *COCT_MT010001UV01* que es el que se definió con nombre *IN1* anteriormente, en el resto de transformaciones se usará *IN2*, *IN3*, etc.

Por último, en la etiqueta *outmodel*, se define la ruta donde se guardará el esquema resultante de la transformación.

```
<atl.launch path="HL7ToXMI.atl">
<option name="allowInterModelReferences" value="true"/>
  <inmodel name="IN" model="IN1"/>
  <inmodel name="HL7" model="HL7"/>
  <inmodel name="PRO" model="PRO" />
  <inmodel name="DT" model="DT" />
  <inmodel name="UML" model="UML2" />
  <outmodel name="OUT" model="OUT1" metamodel="UML2"
   path="./resultados/COCT_MT010000UV01.uml"/>
</atl.launch>
```

Ejemplo 20.3 Definición de la conversión de un esquema en el fichero build.xml

Para ejecutar las transformaciones definidas en ese XML, deben seguirse las siguientes instrucciones: se debe hacer click derecho sobre el fichero XML, y seleccionar la opción de *run as external tools configurations*. A continuación, hay que seleccionar *Run in the same JRE as the workspace* dentro de la pestaña JRE (figura 20.5). Esto último es muy importante, ya que si no se activa esta opción, las transformaciones no se ejecutarán correctamente.

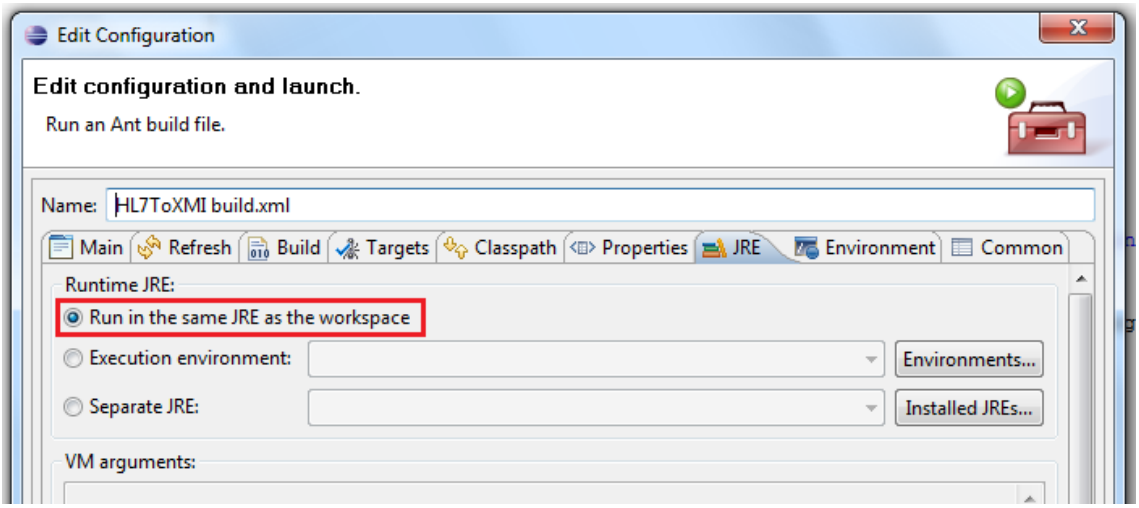


Figura 20.5 Configuración de ejecución para el fichero build.xml

20.3 Refinamiento

Una vez se tienen todos los esquemas convertidos, el último paso consiste en ejecutar la herramienta de refinamiento, contenida dentro del proyecto de *Eclipse* llamado *Post-procesing*.

Para ello, se debe seleccionar nuevamente la opción *run configurations* dentro del menú *run* de *Eclipse* y escoger el archivo Java llamado *Postprocessor*. Esta vez, en la lista de parámetros, lo primero que debemos indicar es si deseamos aplicar el proceso de refinamiento completo o sólo la corrección de enlaces a esquemas externos. Esto es debido a que, tal y como se expuso anteriormente, se ha optado por ofrecer dos versiones de los UML resultantes.

En caso que sólo se quieran corregir los enlaces, el primer parámetro debe ser el *string* "links", en caso contrario, debe ser "completo". Seguidamente, debe introducirse toda la lista de esquemas UML que deben ser refinados, tal y como se muestra en la figura 20.6.

De nuevo, para facilitar esta tarea se proporciona el fichero *HL7ToUml/Post-processing/@listaUML.txt*, que contiene la lista de todos los ficheros UML generados a partir de los ficheros MIF de la versión del estándar escogida.

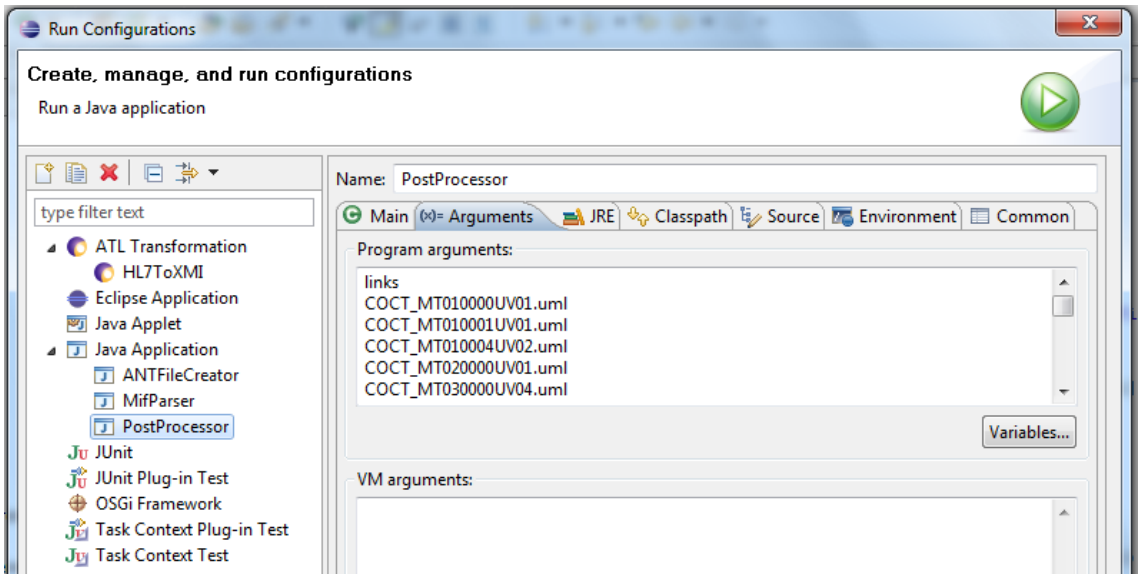


Figura 20.6 Configuración de ejecución de la herramienta de refinamiento

Tras realizar este último paso, si se ha optado por ejecutar la herramienta con la opción de arreglar sólo los enlaces, los UML resultantes se podrán encontrar en el directorio *HL7ToUml/Post-processing/postprocessed_uml(OnlyLinksFixed)*, si por el contrario, se ha

optado por que se trate también el tema de los CMETs, estarán en el directorio *HL7ToUml/Post-processing/postprocessed_uml*.

Llegados a este punto, se dispone del resultado final de la herramienta desarrollada en este proyecto, es decir, de los esquemas conceptuales expresados en UML que representan la información de los esquemas HL7 del estándar.

21 GUÍA DEL DESARROLLADOR

Los esquemas UML generados por la herramienta de conversión, pueden ser aprovechados por los desarrolladores para múltiples fines. Pueden cargarse en herramientas de generación de esquemas gráficos, se pueden emplear en otras que permitan generar código de forma automática, etc.

Una de las herramientas que permite generar esquemas gráficos a partir de los UML obtenidos es *MagicDraw*, como ya se ha podido comprobar en capítulos anteriores de este documento. Para realizar esta tarea, hay que seleccionar la opción *Import from Eclipse UML2 (v3.x) XMI File* del menú *File* (figura 21.1). Seguidamente, se debe escoger el fichero UML que se desea cargar y finalmente, seleccionar la opción *Class Diagram Wizard* dentro del menú *Diagrams* (figura 21.2). Tras ejecutar estos pasos se abre un asistente que permite configurar algunos aspectos del esquema gráfico que se desea generar.

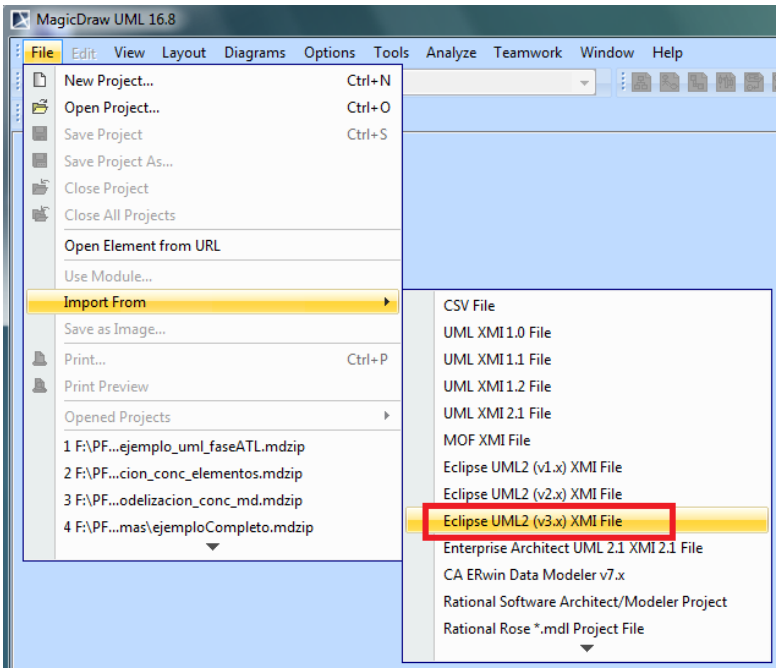


Figura 21.1 Importar un esquema UML desde MagicDraw

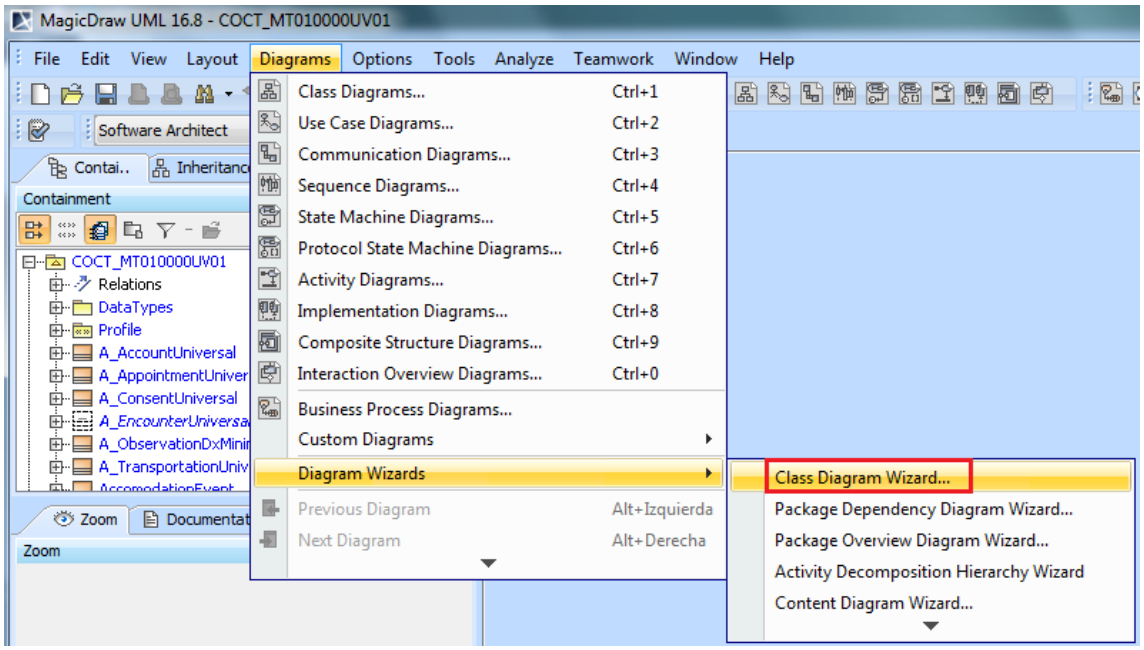


Figura 21.2 Asistente de creación de diagramas de MagicDraw

A continuación, se indica cómo cargar en *Eclipse* los esquemas UML generados por la herramienta de conversión para poder trabajar con ellos de forma programática. Esta plataforma dispone de varias librerías construidas a partir del metamodelo de UML, que nos permiten trabajar con esos esquemas UML utilizando Java de forma muy sencilla, ya que dichas librerías definen funciones para acceder a los valores de los atributos de una clase, a sus asociaciones, etc.

El único requisito para poder ejecutar las acciones descritas en este pequeño tutorial, es tener instalados los *plugins* de *Eclipse* que trabajan con UML. Una forma sencilla de obtenerlos, es bajando la versión *Modeling* de *Eclipse* que incluye por defecto en su instalación todos los *plugins* relativos a la modelización conceptual, ya mencionada en el capítulo anterior.

Lo primero que se necesita es poder generar un esquema UML con el formato que la API de Eclipse requiere para poder trabajar, a partir de un fichero XMI. Para ello, nos valdremos de una clase Java publicada en los foros de desarrollo de la comunidad de *Eclipse*, concretamente en [EclFor]. Dicha clase se muestra en el ejemplo 21.1.

En ella, se pueden encontrar tres funciones principales:

- **La creadora UML2ModelSerializer**: se encarga de invocar a todas las funciones de la API necesarias para poder trabajar con ficheros con extensión *.uml* y *.xmi*.

- **ReadUML2Model**: dada una ruta a un fichero XMI, crea y devuelve su elemento raíz de tipo *Model*. Este tipo, corresponde al tipo *Model* que puede encontrarse en el metamodelo de UML. Recordemos que en todos los esquemas generados por la herramienta de conversión, el elemento principal es siempre de tipo *Model*, por lo que en definitiva, esta función nos devuelve un objeto a partir del cual podemos tratar el esquema entero gracias a la API de *Eclipse*.
- **WriteUML2Model**: realiza la operación inversa a la anterior, es decir, dado un objeto de tipo Model que sea el elemento raíz de un determinado esquema UML, genera un fichero XMI que representa dicho esquema.

```
import java.io.File;
import java.io.IOException;
import java.util.Map;
import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.ecore.resource.Resource;
import org.eclipse.emf.ecore.resource.ResourceSet;
import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl;
import org.eclipse.emf.ecore.xmi.impl.XMIResourceFactoryImpl;
import org.eclipse.uml2.uml.Model;
import org.eclipse.uml2.uml.Package;
import org.eclipse.uml2.uml.UMLPackage;
import org.eclipse.uml2.uml.resource.UMLResource;
public class UML2ModelSerializer
{
    public static UML2ModelSerializer INSTANCE = new
     UML2ModelSerializer();

    private UML2ModelSerializer() {
      Resource.Factory.Registry.INSTANCE.
       getExtensionToFactoryMap().put(UMLResource.FILE_EXTENSION,
       UMLResource.Factory.INSTANCE);
      Resource.Factory.Registry.INSTANCE.
       getExtensionToFactoryMap().put("xmi",
       new XMIResourceFactoryImpl());
      Resource.Factory.Registry.INSTANCE.
       getExtensionToFactoryMap().put("uml",
       new XMIResourceFactoryImpl());
    }

    public Package readUML2Model(String modelFileName) throws
     IOException {
      return readUML2Model(new File(modelFileName));
    }

    public Model readUML2Model(File modelFile) throws IOException {
      ResourceSet resourceSet = new ResourceSetImpl();
      resourceSet.getPackageRegistry().put(UMLPackage.eNS_URI,
       UMLPackage.eINSTANCE);
      resourceSet.getResourceFactoryRegistry().
       getExtensionToFactoryMap().put(UMLResource.FILE_EXTENSION,
       UMLResource.Factory.INSTANCE);
      Map<URI, URI> uriMap =
       resourceSet.getURIConverter().getURIMap();
      URI uri = URI.createURI("jar:file:/C:/eclipse/plugins/
```

```
        org.eclipse.uml2.uml.resources_2.2.0.v200805131030.jar!/");
      uriMap.put(URI.createURI(UMLResource.LIBRARIES_PATHMAP),
       uri.appendSegment("libraries").appendSegment(""));
      uriMap.put(URI.createURI(UMLResource.METAMODELS_PATHMAP),
       uri.appendSegment("metamodels").appendSegment(""));
      uriMap.put(URI.createURI(UMLResource.PROFILES_PATHMAP),
       uri.appendSegment("profiles").appendSegment(""));
      String path = modelFile.getCanonicalPath();
      Resource res =
       resourceSet.getResource(URI.createURI(path),true);
      return (Model)res.getContents().get(0);
    }

    public void writeUML2Model(String modelFileName, Model model)
     throws IOException {
      writeUML2Model(new File(modelFileName), model);
    }

    public void writeUML2Model(File out, Model model) throws
     IOException {
      String path = out.getCanonicalPath();
      Resource resource =
       UMLResource.Factory.INSTANCE.
       createResource(URI.createFileURI(path));
      resource.getContents().add(model);
      resource.save(null);
    }
}
```

Ejemplo 21.1 Clase Java que permite leer esquemas UML y escribir en ellos

A partir de esta clase y de la API de Eclipse, resulta fácil trabajar con los esquemas UML. En el ejemplo 21.2, se muestra una clase Java que imprime por la salida estándar los nombres de todas las clases presentes en el esquema. Es un ejemplo simple, pero a partir de él, se abren múltiples posibilidades para los desarrolladores que deseen trabajar con los esquemas de forma programática.

La función *getAllClasses*, recibe por parámetro un objeto de tipo *Model*, y devuelve la lista de objetos de tipo *Class* que contiene. Es importante destacar que tanto ese tipo *Class* como el *Model*, corresponden a sus elementos homónimos del metamodelo de UML.

Lo más interesante de esta función, es la llamada *model.allOwnedElements()* que se realiza dentro de la condición del bucle. Recordemos que en el metamodelo de UML, se define una relación entre *Model* y *Element* que indica qué elementos están contenidos en un determinado modelo. La función nombrada, precisamente lo que hace es devolver todos los elementos de tipo *Element* que están relacionados con el modelo *model*. Esta llamada, permite que nos hagamos una idea de lo útil que es la API de *Eclipse* mencionada anteriormente, ya que se definen funciones similares para cada uno de los elementos que componen el metamodelo de UML.

Por otro lado, en la función *main*, lo primero que se hace es cargar un modelo, que en este caso es el que tiene por identificador *PORR_MT049007.uml,* mediante la función *readUML2Model* ya explicada. Seguidamente, se define un bucle que recorre todos los elementos de tipo *Class* obtenidos con la función *getAllClasses* ya explicada anteriormente, e imprime sus nombres. En este fragmento de código, lo más destacable es el uso de la función *getName*. Recordemos que en el metamodelo de UML, la clase Class presenta un atributo llamado *name* que es el que devuelve dicha función. De nuevo, aquí se puede apreciar la potencia de la API de *Eclipse* que trabaja con UML. Mediante métodos *get*, es posible acceder a cualquiera de los atributos contenidos en las clases presentes en el metamodelo de UML y con los *set*, poder actualizar cualquiera de los valores presentes en el esquema.

```
public class ListNames {
  private static ArrayList<Class> getAllClasses(Model model) {
    ArrayList<Class> res = new ArrayList<Class>();
    for (Element e : model.allOwnedElements()) {
      if (e instanceof Class) {
        Class c = (Class) e;
        res.add(c);
      }
    }
    return res;
  }

  public static void main(String[] args) throws IOException {
    Model m = (Model)
    UML2ModelSerializer.INSTANCE.
     readUML2Model("PORR_MT049007UV01.uml");
    for (Iterator iterator = getAllClasses(m).iterator();
     iterator.hasNext();) {
      Class c = (Class) iterator.next();
      System.out.println(c.getName());
    }
  }

}
```

Ejemplo 21.2 Clase Java que imprime el nombre de todas las clases presentes en el esquema

22 PRUEBAS DE RENDIMIENTO

El requisito más importante que la herramienta desarrollada debe cumplir, es que los esquemas conceptuales que genere sean correctos. La eficiencia de la aplicación queda en un segundo plano, no obstante, creemos conveniente hacer un pequeño análisis de los tiempos que tarda en realizar las conversiones.

La herramienta de conversión, sólo deberá ejecutarse bien cuando aparezca una nueva versión que corrija errores que puedan existir, o bien cuando se publique una nueva versión de los esquemas del estándar, hecho que ocurre una vez al año.

Las pruebas se han ejecutado en un PC con unos cinco años de antigüedad, un *Pentium 4* con una frecuencia de reloj de 3 Ghz y 2GB de memoria RAM, con lo que en uno actual, es de esperar que los tiempos expuestos a continuación se reduzcan considerablemente.

El *parser* de los ficheros MIF del estándar tarda unos 35 segundos de media en generar los XML de todos los esquemas de la versión 2009 del estándar, por otro lado, ejecutar todas las transformaciones ATL requiere unos 5 minutos y finalmente, la ejecución de la herramienta de refinamiento, se demora alrededor de 25 segundos.

Como conclusión, se puede afirmar que estos números son perfectamente asumibles teniendo en cuenta la poca frecuencia con la que es de esperar que se ejecute el programa.

23 DISEÑO DE LOS ESQUEMAS CON UMLCANVAS

Para ejecutar la herramienta de conversión de modelos desarrollada es necesario, tal y como se ha explicado en la guía del usuario, tener instalado *Eclipse* y sus *plugins* relativos al modelado.

Con el objetivo de eliminar este requisito en cuanto al software necesario y hacer los esquemas conceptuales más accesibles a los usuarios, se ha creído conveniente diseñarlos utilizando la herramienta *UmlCanvas*, concretamente su versión 0.4.

UmlCanvas es una librería escrita en *Javascript* que utiliza HTML5 para construir esquemas conceptuales UML gráficos e interactivos. De esta manera, es posible publicar en una página web toda una serie de esquemas con los que los usuarios pueden interactuar cambiando de lugar las clases, moviendo las asociaciones, etc, prácticamente como si de una aplicación de escritorio se tratase.

UmlCanvas, utiliza una sintaxis propia para definir los elementos UML. Así por ejemplo, el código mostrado en el ejemplo 23.1, genera la clase *Persona* de la figura 23.1.

```
var myClass = new UmlCanvas.Class({ name: "Persona" });
myClass.addAttribute({ name: "dni", type: "String",
       visibility: "private" });
myClass.addAttribute({ name: "nombre", type: "String",
       visibility: "private" });
myClass.addOperation({ name: "matricular",
       arguments: [ { name: "universidad", type: "String" },
       { name: "carrera", type: "String" } ] });
```

Ejemplo 23.1 Definición de una clase en UmlCanvas

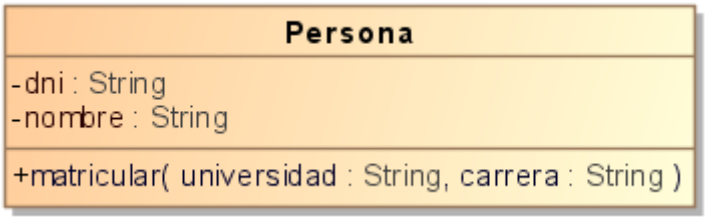


Figura 23.1 Clase Persona en UML

Para convertir los esquemas UML generados por la herramienta de conversión a ese lenguaje propio de *UmlCanvas*, se ha optado por utilizar la API de que dispone *Eclipse* para tratar ficheros UML, que ya ha sido explicada en la guía del desarrollador de este documento y que nos permite tratar de forma programática los esquemas conceptuales.

En la figura 23.2, se encuentra un fragmento del RIM diseñado con la librería *UmlCanvas*. Como se puede comprobar, los esquemas se muestran de una forma visualmente agradable y además, se pueden colorear las clases al igual que se hace en los esquemas del estándar HL7.

Figura 23.2 Fragmento del RIM diseñado con UmlCanvas

PLANIFICACIÓN Y COSTES

24 PLANIFICACIÓN

El proyecto comenzó formalmente el día 13 de diciembre de 2010 cuando fue inscrito. No obstante, es importante recalcar que antes de esa fecha, ya se habían mantenido reuniones con el director y el codirector del proyecto, y se había trabajado principalmente en la parte relacionada con el estudio del estándar HL7 y de las herramientas ya disponibles.

Concretamente, la reunión en la que se habló por primera vez de los objetivos que debería cubrir el proyecto tuvo lugar el día 29 de junio de 2010.

A continuación, se presenta la planificación inicial realizada en los primeros compases del proyecto, y posteriormente la real, es decir, aquella confeccionada una vez terminado y que permite comprobar la cantidad de horas que finalmente se ha dedicado a cada una de las tareas que componen el proyecto.

24.1 Planificación inicial

La planificación inicial, proyectada cuando se inició el proyecto, se puede observar en el Diagrama de Gantt de la figura 24.1.

Para entenderla es importante tener en cuenta que, desde un primer momento se tenía claro que la carga de trabajo no estaría uniformemente distribuida entre los meses de julio de 2010 y de junio de 2011.

Esto es debido a que entre los meses de septiembre de 2010 y enero de 2011, debía cursar las cinco últimas asignaturas de la carrera que me quedaban, y por tanto, durante ese periodo me resultaría imposible dedicarme plenamente al proyecto.

De este modo, a grandes rasgos, la planificación consistía en dedicar la época vacacional de verano a la familiarización con el estándar HL7, en el último cuatrimestre de asignaturas ir avanzando lo posible para no perder el contacto con el proyecto, y realizar el grueso del trabajo desde el mes de febrero de 2010, fecha a partir de la cual podría disponer de una dedicación casi exclusiva. Finalmente, durante el mes de enero, no se planificaron tareas, debido a que tenía que examinarme de las últimas asignaturas de la carrera.

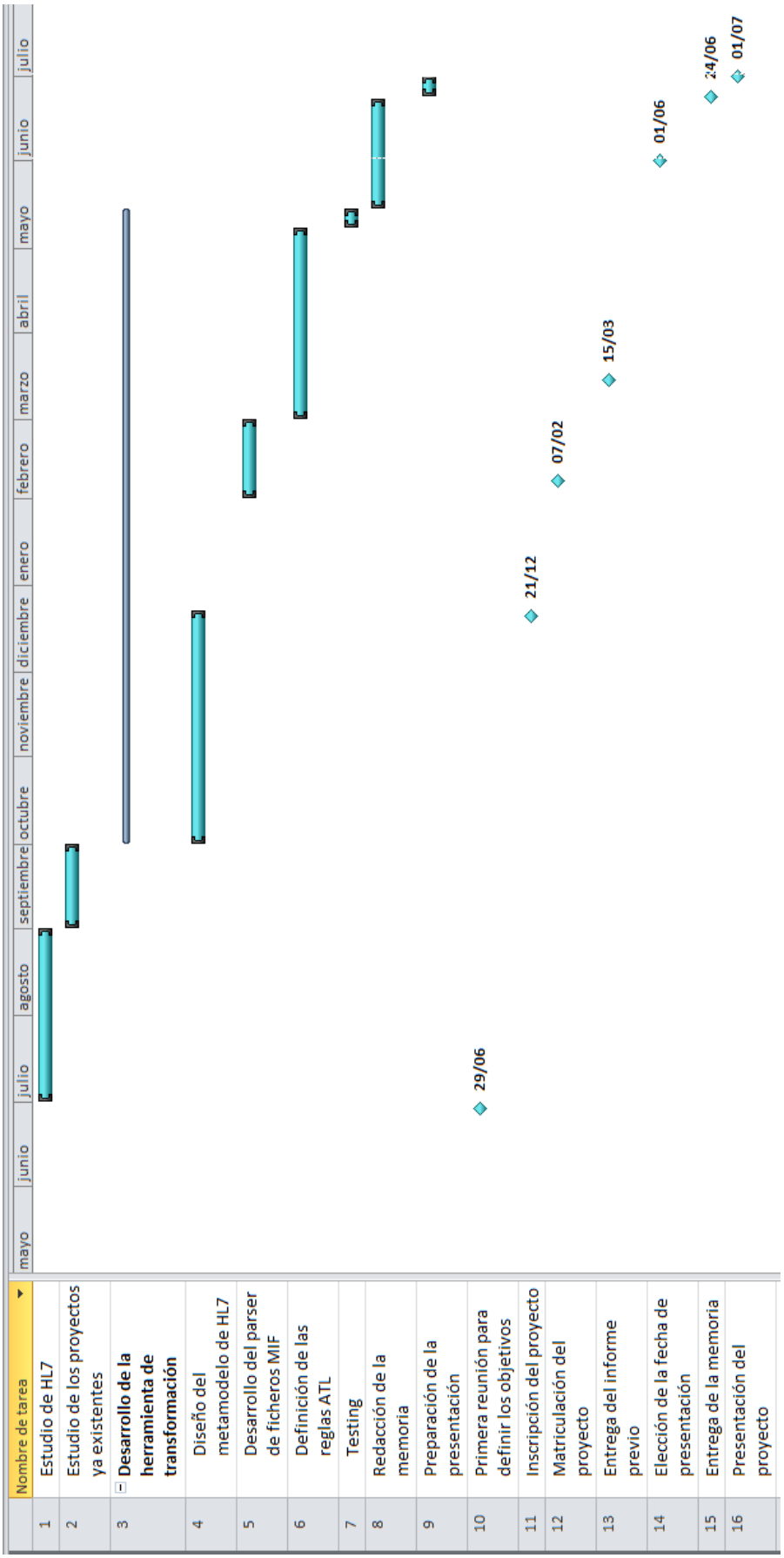


Figura 24.1 Diagrama de Gantt de la planificación inicial

En el diagrama también pueden apreciarse los hitos del proyecto, que representan las fechas límites impuestas por la normativa de la universidad. No son fechas fijas, y por tanto, podrían adelantarse si se diesen las condiciones necesarias para ello durante el desarrollo del proyecto.

Como la dedicación no es uniforme, es preciso complementar el diagrama de Gantt mostrado anteriormente, con una tabla que indica el número de horas estimadas para llevar a cabo cada una de las tareas que aparecen (tabla 24.1).

Tarea	Dedicación (horas)
Estudio del estándar HL7	60
Estudio de los proyectos ya existentes que trabajan con HL7	30
Diseño del metamodelo de HL7	70
Desarrollo del *parser* de ficheros MIF	110
Definición de las reglas ATL	210
Testing	40
Redacción de la memoria	200
Preparación de la presentación	30
	Total: **750**

Tabla 24.1 Número de horas dedicadas a cada tarea según la planificación inicial

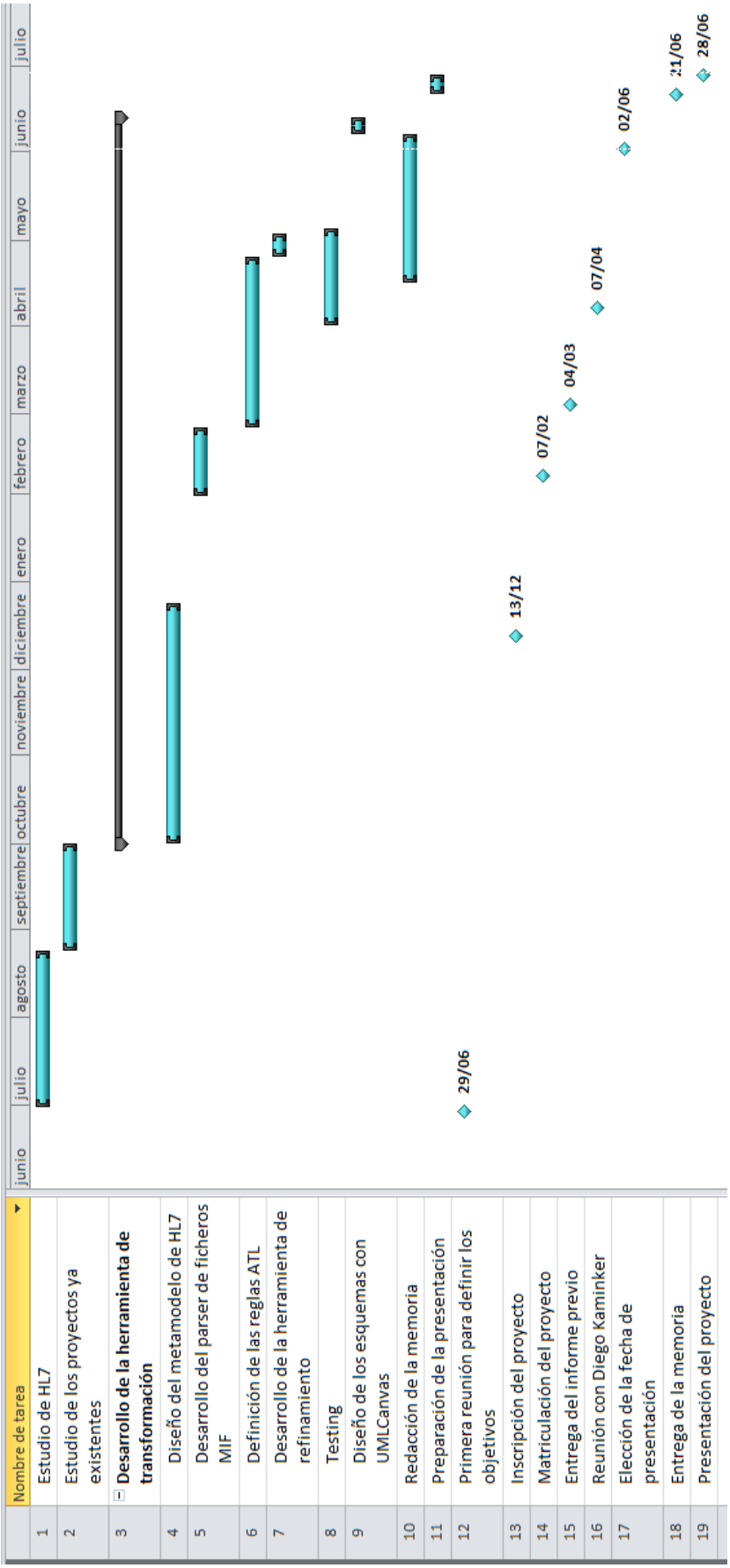
24.2 Planificación real

La planificación real se muestra en el diagrama de Gantt de la figura 24.2.

Como se puede observar, la distribución de las tareas a lo largo del tiempo se ha mantenido sin cambios importantes.

La principal diferencia respecto a la planificación inicial es la introducción de dos nuevas tareas que no se previeron.

La primera es la del refinamiento de los esquemas UML obtenidos a partir de las reglas ATL, ya que en un primer momento no se contempló la posibilidad de que los esquemas resultado de ejecutar las reglas ATL no contasen con todas las características definidas.



	Nombre de tarea
1	Estudio de HL7
2	Estudio de los proyectos ya existentes
3	**Desarrollo de la herramienta de transformación**
4	Diseño del metamodelo de HL7
5	Desarrollo del parser de ficheros MIF
6	Definición de las reglas ATL
7	Desarrollo de la herramienta de refinamiento
8	Testing
9	Diseño de los esquemas con UMLCanvas
10	Redacción de la memoria
11	Preparación de la presentación
12	Primera reunión para definir los objetivos
13	Inscripción del proyecto
14	Matriculación del proyecto
15	Entrega del informe previo
16	Reunión con Diego Kaminker
17	Elección de la fecha de presentación
18	Entrega de la memoria
19	Presentación del proyecto

Figura 24.2 Diagrama de Gantt de la planificación real

La segunda es el diseño de la parte gráfica de los esquemas utilizando la herramienta *UMLCanvas*, ya que se pensó en una manera más fácil de poder acceder a los esquemas generados por la herramienta de conversión desarrollada, cuando el proyecto ya estaba bastante avanzando.

De nuevo, se hace necesario mostrar las horas empleadas en cada una de las tareas para completar la información del diagrama de Gantt (tabla 24.2).

Tarea	Dedicación (horas)
Estudio del estándar HL7	60
Estudio de los proyectos ya existentes que trabajan con HL7	30
Diseño del metamodelo de HL7	80
Desarrollo del *parser* de ficheros MIF	130
Definición de las reglas ATL	200
Desarrollo de la herramienta de refinamiento	40
Diseño de los esquemas usando *UMLCanvas*	40
Testing	40
Redacción de la memoria	220
Preparación de la presentación	40
	Total: **880**

Tabla 24.2 Número de horas dedicadas a cada tarea según la planificación real

Como se puede comprobar, no hay grandes desviaciones entre la cantidad de horas previstas y las que finalmente se han destinado a realizar cada una de las tareas. El aumento del tiempo necesario para completar el proyecto que se da en la planificación real viene dado, principalmente, por el hecho de haber incluido las dos tareas ya comentadas que no se planificaron al inicio.

25 COSTE

Con el objetivo de poder hacernos una idea del coste de este proyecto, en este capítulo se presenta un pequeño estudio basado en la cantidad de horas necesarias para realizarlo, según lo descrito en la planificación real.

Para poder realizar una aproximación más realista, es importante tener en cuenta que en este proyecto, intervienen dos perfiles profesionales distintos, el analista y el programador, que tienen sueldos diferentes.

En cada una de las tareas mostradas anteriormente, pueden intervenir los dos y con una carga distinta de trabajo. Así por ejemplo, en la definición de las reglas ATL, es obvio que se trata de programarlas utilizando el lenguaje propio de la herramienta y que por tanto, el grueso del trabajo corresponde a un programador. No obstante, no hay que olvidar, que antes alguien ha tenido que decidir en qué se transformará cada uno de los elementos de HL7 y haber investigado qué es lo que hacen las herramientas de conversión ya existentes, y eso es un trabajo más propio de un analista.

Por ello, el primer paso para calcular el coste de este proyecto, consiste en saber cuántas horas de trabajo corresponden a cada uno de esos dos perfiles. Para ello, tal y como se muestra en la tabla 25.1, lo que se debe hacer es calcular el número de horas que les corresponden en cada tarea a partir de su dedicación.

Tarea	Dedicación total (horas)	Dedicación analista (horas)	Dedicación programador (horas)
Estudio del estándar HL7	60	48 (80%)	12 (20%)
Estudio de los proyectos ya existentes que trabajan con HL7	30	30 (100%)	0 (0%)
Diseño del metamodelo de HL7	80	56 (70%)	24 (30%)
Desarrollo del *parser* de ficheros MIF	130	32.5 (25%)	97.5 (75%)
Definición de las reglas ATL	200	50 (25%)	150 (75%)
Desarrollo de la herramienta de refinamiento	40	4 (10%)	36 (90%)
Diseño de los esquemas usando *UMLCanvas*	40	0 (0%)	40 (100%)

Testing	40	0 (0%)	40 (100%)
Redacción de la memoria	220	176 (80%)	44 (20%)
Preparación de la presentación	40	40 (100%)	0 (0%)
	Total: **880**	Total: **436.5 (49.6%)**	Total: **443.5 (50.4%)**

Tabla 25.1 Dedicación correspondiente a cada uno de los perfiles

Tras haber calculado el número de horas correspondiente a cada uno de los perfiles, el siguiente paso consiste en multiplicarlo por el sueldo de cada uno de ellos para obtener el coste total.

Según [IJsal], el salario anual medio de un programador en España actualmente es de unos 26000€ y el de un analista está alrededor de los 30000€.

Suponiendo jornadas laborales de 8 horas, trabajando 23 días al mes y con vacaciones pagadas, para obtener el salario por hora, deberemos dividir las dos cantidades obtenidas anteriormente entre 8 x 23 x 11 = 2024. De este modo, se obtiene que el sueldo medio por hora de un programador es de unos 12.85€, mientras que el de un analista es de 14.82€.

Llegados a este punto, ya se tienen todos los datos necesarios para calcular el coste del proyecto asociado a los trabajadores. El resultado se muestra en la tabla 25.2.

Perfil	Horas de trabajo	Sueldo €/h	Coste
Analista	436.5	14.82	6468.93€
Programador	443.5	12.85	5698.98€
			Total: **12167.91€**

Tabla 25.2 Cálculo del coste del proyecto

En este proyecto, todo el software utilizado es gratuito, con lo que el coste no se ve incrementado en este aspecto.

Por otro lado, en cuanto a los costes del hardware, debido a que no necesita cumplir ningún requisito especial, y considerando que un ordenador actual dura unos cuatro años, el resultado de dividir su precio entre las horas de trabajo que proporciona en ese tiempo y multiplicarlas por las dedicadas en este proyecto sería despreciable.

Así, el coste total aproximado del proyecto sería de unos 12167€.

CONCLUSIONES

26 CONCLUSIONES

Para cerrar el documento, en este capítulo se describen las conclusiones más importantes que se desprenden de la realización del presente proyecto.

En primer lugar, se presentan aquellas que se pueden extraer del proyecto en sí, seguidamente, aquellas que hacen referencia a los conocimientos y capacidades adquiridas, a continuación, se exponen toda una serie de posibles mejoras y ampliaciones de la herramienta construida, y finalmente, se ofrece una valoración personal del proyecto en general.

26.1 Acerca del proyecto

- Los objetivos propuestos al inicio del proyecto, se han cumplido. La herramienta de conversión desarrollada es capaz de transformar los esquemas conceptuales del estándar HL7 a UML.

- Todas las conversiones realizadas se han justificado debidamente, a diferencia de lo que ocurre con las herramientas existentes. Además, se han apuntado las razones que hacen que los resultados arrojados por algunas de ellas sean incorrectos y se han convertido todos los esquemas conceptuales disponibles. Por ello, se está en disposición de afirmar que el software construido mejora en muchos aspectos a los ya existentes y constituye una nueva alternativa seria y a tener en cuenta para aquellas personas que deseen trabajar en UML con los esquemas de HL7.

- A pesar de que el software desarrollado ofrece una solución que incluso algunos miembros de HL7 pedían, es pronto para determinar si el proyecto será un éxito o no. Aún se debe dar a conocer la herramienta y esperar un tiempo prudencial para ver qué grado de aceptación tiene y qué tipo de críticas despierta entre las personas próximas al estándar HL7.

26.2 Acerca de los conocimientos y capacidades adquiridas

- El haber realizado un proyecto de un tamaño considerable, me ha permitido mejorar mis capacidades de organización, planificación y resolución de problemas no previstos, al mismo tiempo que ha incrementado mi seguridad de cara a afrontar futuros proyectos.

- En este proyecto, he tenido la posibilidad de trabajar con un estándar, el HL7, y de poder analizar tanto sus puntos fuertes así como los débiles. Esto me resultará de ayuda en un futuro, puesto que es de esperar que el número de estándares con los que un ingeniero informático tenga que tratar a lo largo de su carrera profesional sea considerable.

- He podido poner en práctica a la vez que expandir, los conocimientos teóricos aprendidos en varias asignaturas de la carrera, en especial los de aquellas enmarcadas dentro del campo de la ingeniería del software. Sin disponer de esa base teórica, no habría sido posible llevar a cabo el presente proyecto.

- He podido trabajar con una plataforma potente como lo es *Eclipse* y aprender que gracias a algunas herramientas como ésa, resulta posible aplicar la máxima de no reinventar la rueda. Este entorno provee a los programadores de un amplísimo abanico de APIs y utilidades capaces de realizar funciones que no conocía al inicio de este proyecto.

- El código del proyecto resulta más fácil de mantener gracias al hecho de haber diseñado el software siguiendo una arquitectura que separa claramente las distintas partes que componen el proyecto. Al tener que corregir errores o añadir nuevas funcionalidades, he podido darme cuenta de lo importante que resulta este hecho de cara a facilitar el trabajo futuro que potencialmente pueda dedicarse en la herramienta.

26.3 Posibles ampliaciones y mejoras

- Muchos de los cambios que deban incorporarse a la herramienta de conversión, dependen directamente de decisiones que se tomen desde la organización HL7. El hecho que introduzcan nuevos elementos, o que cambien la estructura de los ficheros fuente que contienen la información de los esquemas, provocará irremediablemente que se tengan que realizar algunos ajustes en la herramienta si se quiere preservar la compatibilidad con las nuevas versiones del estándar. Así por ejemplo, desde HL7 se está trabajando en una nueva versión de la especificación de los tipos de datos, la cual aún no es lo suficientemente madura, pero cuando alcance un nivel de estabilidad razonable con toda probabilidad se empezará a incorporar en los esquemas conceptuales.

- Actualmente, para ejecutar la herramienta, es necesario tener instalado Eclipse y toda una serie de *plugins* de modelización conceptual. Sería buena idea disponer de un servicio web desde el que se pudiesen ejecutar para poder eliminar estos requisitos de software.

- El dotar a la herramienta de una interfaz gráfica mejoraría su usabilidad.

26.4 Valoración personal

Bajo mi punto de vista, una de las cosas más destacables acerca de la realización de este proyecto es que surge para cubrir una necesidad real y que puede ser usado en un futuro para complementar lo que ofrece el estándar HL7. Como ya se ha comentado anteriormente, desde diversas páginas dedicadas a HL7 así como desde sus listas de correo, se alzan algunas opiniones pidiendo una versión UML de los esquemas conceptuales.

El que un proyecto propio pueda ser utilizado por otras personas, supone siempre un hecho motivador. Evidentemente, para juzgar si el proyecto ha resultado ser un éxito, hay que esperar a ver qué tipo de reacciones se producen entre las personas que trabajan con el estándar y para ello aún es pronto, puesto que en el momento de escribir estas líneas, el proyecto aún no se ha dado a conocer. No obstante, nuestras expectativas al respecto son optimistas.

Por otro lado, tal y como se desprende de las conclusiones anteriormente expuestas, personalmente este proyecto me ha permitido tanto ampliar mis conocimientos sobre la disciplina de la ingeniería del software, así como mejorar algunas de las capacidades imprescindibles en el desarrollo de cualquier proyecto.

Para concluir este capítulo, me gustaría apuntar que por todo lo expuesto anteriormente, la experiencia de realizar este proyecto ha resultado positiva y enriquecedora.

ANEXOS

ANEXO A. Definición del metamodelo de HL7 en USE

```
model HL7

-- enumerations
enum CStrength {CWE, CNE}
enum noteType {Definition, Usage}

-- classes

class ActRelationshipType < Class
end

class ActType < Class
end

class Association < Element
end

class Choice < ChoosableElement
end

abstract class ChoosableElement < Type
 operations
  conformsTo(ce:ChoosableElement):Boolean =
    if (self.oclIsTypeOf(CMET)) then
      self.oclAsType(CMET).mainElement.conformsTo(ce)
    else if (self.oclIsTypeOf(Choice) and not
     self.oclAsType(Choice).ownedElement->exists(oe | oe = self))
     then
      self.oclAsType(Choice).ownedElement->
      forAll(oe|oe.conformsTo(ce))
    else if (self.oclIsTypeOf(EntityType)) then
      ce.oclIsTypeOf(EntityType)
      or (ce.oclIsTypeOf(CMET) and
      ce.oclAsType(CMET).mainElement.conformsTo(self))
      or (ce.oclIsTypeOf(Choice) and (not
      ce.oclAsType(Choice).ownedElement->exists(oe | oe = ce)) and
      ce.oclAsType(Choice).ownedElement->
      forAll(coe|coe.conformsTo(self)))
    else if (self.oclIsTypeOf(RoleLinkType)) then
      ce.oclIsTypeOf(RoleLinkType)
      or (ce.oclIsTypeOf(CMET) and
      ce.oclAsType(CMET).mainElement.conformsTo(self))
      or (ce.oclIsTypeOf(Choice) and (not
      ce.oclAsType(Choice).ownedElement->exists(oe | oe = ce)) and
      ce.oclAsType(Choice).ownedElement->
      forAll(coe|coe.conformsTo(self)))
    else if (self.oclIsTypeOf(RoleType)) then
      ce.oclIsTypeOf(RoleType)
      or (ce.oclIsTypeOf(CMET) and
      ce.oclAsType(CMET).mainElement.conformsTo(self))
      or (ce.oclIsTypeOf(Choice) and (not
      ce.oclAsType(Choice).ownedElement->exists(oe | oe = ce)) and
      ce.oclAsType(Choice).ownedElement->
      forAll(coe|coe.conformsTo(self)))
    else if (self.oclIsTypeOf(ParticipationType)) then
      ce.oclIsTypeOf(ParticipationType)
      or (ce.oclIsTypeOf(CMET) and
      ce.oclAsType(CMET).mainElement.conformsTo(self))
```

```
      or (ce.oclIsTypeOf(Choice) and (not
      ce.oclAsType(Choice).ownedElement->exists(oe | oe = ce)) and
      ce.oclAsType(Choice).ownedElement->
      forAll(coe|coe.conformsTo(self)))
    else if (self.oclIsTypeOf(ActType)) then
      ce.oclIsTypeOf(ActType)
      or (ce.oclIsTypeOf(CMET) and
      ce.oclAsType(CMET).mainElement.conformsTo(self))
      or (ce.oclIsTypeOf(Choice) and (not
      ce.oclAsType(Choice).ownedElement->exists(oe | oe = ce)) and
      ce.oclAsType(Choice).ownedElement->
      forAll(coe|coe.conformsTo(self)))
    else if (self.oclIsTypeOf(ActRelationshipType)) then
      ce.oclIsTypeOf(ActRelationshipType)
      or (ce.oclIsTypeOf(CMET) and
      ce.oclAsType(CMET).mainElement.conformsTo(self))
      or (ce.oclIsTypeOf(Choice) and (not
      ce.oclAsType(Choice).ownedElement->exists(oe | oe = ce)) and
      ce.oclAsType(Choice).ownedElement->
      forAll(coe|coe.conformsTo(self)))
    else if (self.oclIsTypeOf(InfraestructureType)) then
      ce.oclIsTypeOf(InfraestructureType)
      or (ce.oclIsTypeOf(CMET) and
      ce.oclAsType(CMET).mainElement.conformsTo(self))
      or (ce.oclIsTypeOf(Choice) and (not
      ce.oclAsType(Choice).ownedElement->exists(oe | oe = ce)) and
      ce.oclAsType(Choice).ownedElement->
      forAll(coe|coe.conformsTo(self)))
    else false
    endif
    endif
    endif
    endif
    endif
    endif
    endif
    endif
    endif
end

abstract class Class < ChoosableElement
end

class CMET < ChoosableElement
attributes
  rootClassCode: String
  attributionLevel: String
  identifier: String
  mainElementTypeCode: String
end

class Constraint < Element
attributes
  body: String
end

abstract class DataType < Type
end

abstract class Element
end
```

```
class EntityType < Class
end

class EntryPoint < NamedElement
attributes
  identifier: String
  description: String
end

class Generalization < Element
end

class InfraestructureType < Class
end

abstract class MultiplicityElement < Element
attributes
  upper: Integer
  lower: Integer
end

abstract class NamedElement < Element
attributes
  name: String
end

class Note < Element
attributes
  body: String
  type: noteType
end

class ParticipationType < Class
end

class Property < MultiplicityElement, TypedElement
attributes
  domainName: String
  codeSystemName: String
  mnemonic: String
  defaultValue: String
  codingStrength: CStrength
end

class RoleLinkType < Class
end

class RoleType < Class
end

abstract class Type < NamedElement
end

abstract class TypedElement < NamedElement
end

abstract class ValueSpecification < TypedElement
end

-- associations
```

```
association ClassToOwnedAttributeofProperty between
  Class[0..1]
  Property[*] role ownedAttribute
end

association memberEndOfPropertyToAssociation between
  Property[2] role memberEnd
  Association[0..1]
end

association owningPropertyOfPropertyToDefaultValueOfSpecification
between
  Property[0..1] role owningProperty
  ValueSpecification[0..1] role defaultValue
end

association childOfClassToGeneralization between
  Class[1] role child
  Generalization[*]
end

association parentOfClassToSpecializationOfGeneralization between
  Class[1] role parent
  Generalization[*] role specialization
end

association ElementToNote between
  Element[*]
  Note[*]
end

association ChoiceToOwnedElementOfChoosableElement between
  Choice[*]
  ChoosableElement[2..*] role ownedElement
end

association ownerOfCMETToOwnedElementOfChoosableElement between
  CMET[*] role owner
  ChoosableElement[1..*] role ownedElement
end

association CMETToMainElementOfChoosableElement between
  CMET[*]
  ChoosableElement[1] role mainElement
end

association EntryPointToOwnedElementOfChoosableElement between
  EntryPoint[0..1]
  ChoosableElement[*] role ownedElement
end

association TypeToTypedElement between
  Type[0..1]
  TypedElement[*]
end

association NamedElementToOwnedRuleOfConstraint between
  NamedElement[0..1]
  Constraint[*] role ownedRule
end
```

```
association ConstraintToConstrainedElementOfConstraint between
  Constraint[*]
  Element[*] role ConstrainedElement
end


-- constraints

constraints

context CMET

--Un CMET no se puede incluir a sí mismo.
inv CMETNoSeIncluyeASiMismo:
  not self.ownedElement->exists(oe | oe = self)

--El elemento principal de un CMET tiene que estar entre los
elementos que incluye
inv CMETIncluyeSuElementoPrincipal:
  self.ownedElement->exists(oe | oe = self.mainElement)


context Choice

--Un Choice no se puede incluir a sí mismo.
inv ChoiceNoSeIncluyeASiMismo:
  not self.ownedElement->exists(oe | oe = self)

--Un Choice sólo incluye elementos compatibles entre sí
inv ChoiceIncluyeElementosCompatibles:
  self.ownedElement->
  forAll(ce1,ce2:ChoosableElement|ce1.conformsTo(ce2))


context Association

--Si un miembro de la asociación es una clase EntityType, el otro
tiene que ser una clase RoleType o InfraestructureType
inv UnaEntitySeRelacionaConUnRoleoConUnInfraestructure:
  (self.memberEnd->exists(me | me.type.oclIsTypeOf(EntityType)))
  implies(self.memberEnd->exists(me | me.type.oclIsTypeOf(RoleType)
  or me.type.oclIsTypeOf(InfraestructureType)
  or (me.type.oclIsTypeOf(Choice) and
  (me.type.oclAsType(Choice).ownedElement->forAll(oe |
  oe.oclIsTypeOf(RoleType) or oe.oclIsTypeOf(InfraestructureType))))
  or (me.type.oclIsTypeOf(CMET) and
  (me.type.oclAsType(CMET).mainElement.oclIsTypeOf(RoleType) or
  me.type.oclAsType(CMET).mainElement.
  oclIsTypeOf(InfraestructureType)))))

--Si un miembro de la asociación es una clase RoleLinkType, el otro
tiene que ser una clase RoleType
inv UnRoleLinkSeRelacionaConUnRole:
  (self.memberEnd->exists(me | me.type.oclIsTypeOf(RoleLinkType)))
  implies (self.memberEnd->exists(me | me.type.oclIsTypeOf(RoleType)
  or (me.type.oclIsTypeOf(Choice) and
  (me.type.oclAsType(Choice).ownedElement->forAll(oe |
  oe.oclIsTypeOf(RoleType))))
  or (me.type.oclIsTypeOf(CMET) and
  (me.type.oclAsType(CMET).mainElement.oclIsTypeOf(RoleType)))))
```

```
--Si un miembro de la asociación es una clase ParticipationType, el
otro tiene que ser una clase RoleType o ActType
inv UnaParticipationSeRelacionaConUnRoleoConUnAct:
  (self.memberEnd->exists(me |
  me.type.oclIsTypeOf(ParticipationType))) implies
  (self.memberEnd->exists(me | me.type.oclIsTypeOf(RoleType) or
  me.type.oclIsTypeOf(ActType)
  or (me.type.oclIsTypeOf(Choice) and
  (me.type.oclAsType(Choice).ownedElement->forAll(oe |
  oe.oclIsTypeOf(RoleType) or oe.oclIsTypeOf(ActType))))
  or (me.type.oclIsTypeOf(CMET) and
  (me.type.oclAsType(CMET).mainElement.oclIsTypeOf(RoleType) or
  me.type.oclAsType(CMET).mainElement.oclIsTypeOf(ActType)))))

--Si un miembro de la asociación es una clase ActRelationshipType, el
otro tiene que ser una clase ActType
inv UnaActRelationshipSeRelacionaConUnAct:
  (self.memberEnd->exists(me |
  me.type.oclIsTypeOf(ActRelationshipType))) implies
  (self.memberEnd->exists(me | me.type.oclIsTypeOf(ActType)
  or (me.type.oclIsTypeOf(Choice) and
  (me.type.oclAsType(Choice).ownedElement->forAll(oe |
  oe.oclIsTypeOf(ActType))))
  or (me.type.oclIsTypeOf(CMET) and
  (me.type.oclAsType(CMET).mainElement.oclIsTypeOf(ActType)))))

--Si un miembro de la asociación es una clase ActType, el otro tiene
que ser una clase ActRelationshipType o ParticipationType
inv UnActSeRelacionaConUnaActRelationshipoConUnaParticipation:
  (self.memberEnd->exists(me | me.type.oclIsTypeOf(ActType))) implies
  (self.memberEnd->exists(me |
  me.type.oclIsTypeOf(ActRelationshipType)
  or me.type.oclIsTypeOf(ParticipationType)
  or (me.type.oclIsTypeOf(Choice) and
  (me.type.oclAsType(Choice).ownedElement->forAll(oe |
  oe.oclIsTypeOf(ActRelationshipType) or
  oe.oclIsTypeOf(ParticipationType))))
  or (me.type.oclIsTypeOf(CMET) and
  (me.type.oclAsType(CMET).mainElement.
  oclIsTypeOf(ActRelationshipType) or
  me.type.oclAsType(CMET).mainElement.
  oclIsTypeOf(ParticipationType)))))

--Si un miembro de la asoación es una clase RoleType, el otro tiene
que ser una clase EntityType, RoleLinkType o ParticipationType
inv UnRoleSeRelacionaConUnaEntityoConUnRoleLinkoConUnaParticipation:
  (self.memberEnd->exists(me | me.type.oclIsTypeOf(RoleType)))
  implies
  (self.memberEnd->exists(me | me.type.oclIsTypeOf(EntityType) or
  me.type.oclIsTypeOf(RoleLinkType) or
  me.type.oclIsTypeOf(ParticipationType)
  or (me.type.oclIsTypeOf(Choice) and
  (me.type.oclAsType(Choice).ownedElement->forAll(oe |
  oe.oclIsTypeOf(EntityType) or oe.oclIsTypeOf(RoleLinkType) or
  oe.oclIsTypeOf(ParticipationType))))
  or (me.type.oclIsTypeOf(CMET) and
  (me.type.oclAsType(CMET).mainElement.oclIsTypeOf(EntityType) or
  me.type.oclAsType(CMET).mainElement.oclIsTypeOf(RoleLinkType) or
  me.type.oclAsType(CMET).mainElement.
  oclIsTypeOf(ParticipationType)))))
```

```
--Si un miembro de la asoación es una clase InfraestructureType, el
otro tiene que ser una clase Entity
inv UnaInfraestructureSeRelacionaConUnaEntity:
  (self.memberEnd->exists(me |
  me.type.oclIsTypeOf(InfraestructureType))) implies
  (self.memberEnd->exists(me | me.type.oclIsTypeOf(EntityType)
  or (me.type.oclIsTypeOf(Choice) and
  (me.type.oclAsType(Choice).ownedElement->forAll(oe |
  oe.oclIsTypeOf(EntityType))))
  or (me.type.oclIsTypeOf(CMET) and
  (me.type.oclAsType(CMET).mainElement.oclIsTypeOf(EntityType)))))
```

ANEXO B. Instanciación USE de un subconjunto del esquema Clinical Document Architecture

```
!create ClinicalDocument : ActType
!create Author : ParticipationType
!create ClinicalDocumentAuthor : Association
!create AuthorProperty1 : Property
!create ClinicalDocumentProperty1 : Property
!insert (ClinicalDocument,ClinicalDocumentProperty1) into
  TypeToTypedElement
!insert (Author,AuthorProperty1) into TypeToTypedElement
!insert (AuthorProperty1,ClinicalDocumentAuthor) into
  memberEndOfPropertyToAssociation
!insert (ClinicalDocumentProperty1,ClinicalDocumentAuthor) into
  memberEndOfPropertyToAssociation
!create Custodian : ParticipationType
!create CustodianProperty1 : Property
!create ClinicalDocumentProperty2 : Property
!insert (ClinicalDocument,ClinicalDocumentProperty2) into
  TypeToTypedElement
!insert (Custodian,CustodianProperty1) into TypeToTypedElement
!create ClinicalDocumentCustodian : Association
!insert (ClinicalDocumentProperty2,ClinicalDocumentCustodian) into
  memberEndOfPropertyToAssociation
!insert (CustodianProperty1,ClinicalDocumentCustodian) into
  memberEndOfPropertyToAssociation
!create AssignedAuthor : RoleType
!create AssignedAuthorProperty1 : Property
!create AuthorAssignedAuthor : Association
!insert (AssignedAuthor,AssignedAuthorProperty1) into
  TypeToTypedElement
!create AuthorProperty2 : Property
!insert (Author,AuthorProperty2) into TypeToTypedElement
!insert (AssignedAuthorProperty1,AuthorAssignedAuthor) into
  memberEndOfPropertyToAssociation
!insert (AuthorProperty2,AuthorAssignedAuthor) into
  memberEndOfPropertyToAssociation
!create Organization : EntityType
!create OrganizationProperty1 : Property
!create AssignedAuthorOrganization : Association
!create AssignedAuthorProperty2 : Property
!insert (AssignedAuthor,AssignedAuthorProperty2) into
  TypeToTypedElement
!insert (Organization,OrganizationProperty1) into TypeToTypedElement
!insert (AssignedAuthorProperty2,AssignedAuthorOrganization) into
  memberEndOfPropertyToAssociation
!insert (OrganizationProperty1,AssignedAuthorOrganization) into
  memberEndOfPropertyToAssociation
!create AuthorChoice : Choice
!create Person : EntityType
!create AuthoringDevice : EntityType
!insert (AuthorChoice,AuthoringDevice) into
  ChoiceToOwnedElementOfChoosableElement
!insert (AuthorChoice,Person) into
  ChoiceToOwnedElementOfChoosableElement
!create AuthorChoiceProperty1 : Property
!insert (AuthorChoice,AuthorChoiceProperty1) into TypeToTypedElement
!create AssignedAuthorAuthorChoice : Association
!create AssignedAuthorProperty3 : Property
```

```
!insert (AssignedAuthor,AssignedAuthorProperty3) into
  TypeToTypedElement
!insert (AssignedAuthorProperty3,AssignedAuthorAuthorChoice) into
  memberEndOfPropertyToAssociation
!insert (AuthorChoiceProperty1,AssignedAuthorAuthorChoice) into
  memberEndOfPropertyToAssociation
!create MaintainedEntity : RoleType
!create MaintainedEntityProperty1 : Property
!create AuthoringDeviceProperty1 : Property
!create MaintainedEntityAuthoringDevice : Association
!insert (AuthoringDevice,AuthoringDeviceProperty1) into
  TypeToTypedElement
!insert (MaintainedEntity,MaintainedEntityProperty1) into
  TypeToTypedElement
!insert (AuthoringDeviceProperty1,MaintainedEntityAuthoringDevice)
  into memberEndOfPropertyToAssociation
!insert (MaintainedEntityProperty1,MaintainedEntityAuthoringDevice)
  into memberEndOfPropertyToAssociation
!create PersonProperty1 : Property
!insert (Person,PersonProperty1) into TypeToTypedElement
!create MaintainedEntityProperty2 : Property
!create MaintainedEntityPerson : Association
!insert (MaintainedEntity,MaintainedEntityProperty2) into
  TypeToTypedElement
!insert (MaintainedEntityProperty2,MaintainedEntityPerson) into
  memberEndOfPropertyToAssociation
!insert (PersonProperty1,MaintainedEntityPerson) into
  memberEndOfPropertyToAssociation
!create CustodianProperty2 : Property
!create AssignedCustodian : RoleType
!create AssignedCustodianProperty1 : Property
!create CustodianAssignedCustodian : Association
!insert (AssignedCustodian,AssignedCustodianProperty1) into
  TypeToTypedElement
!insert (Custodian,CustodianProperty2) into TypeToTypedElement
!insert (CustodianProperty2,CustodianAssignedCustodian) into
  memberEndOfPropertyToAssociation
!insert (AssignedCustodianProperty1,CustodianAssignedCustodian) into
  memberEndOfPropertyToAssociation
!create CustodianOrganization : EntityType
!create CustodianOrganizationProperty1 : Property
!create AssignedCustodianProperty2 : Property
!create AssignedCustodianCustodianOrganization : Association
!insert (AssignedCustodian,AssignedCustodianProperty2) into
  TypeToTypedElement
!insert (CustodianOrganization,CustodianOrganizationProperty1) into
  TypeToTypedElement
!insert (AssignedCustodianProperty2,
  AssignedCustodianCustodianOrganization) into
  memberEndOfPropertyToAssociation
!insert (CustodianOrganizationProperty1,
  AssignedCustodianCustodianOrganization) into
  memberEndOfPropertyToAssociation
!create RecordTarget : ParticipationType
!create RecordTargetProperty1 : Property
!create ClinicalDocumentProperty3 : Property
!insert (ClinicalDocument,ClinicalDocumentProperty3) into
  TypeToTypedElement
!insert (RecordTarget,RecordTargetProperty1) into TypeToTypedElement
!create ClinicalDocumentRecordTarget : Association
!insert (ClinicalDocumentProperty3,ClinicalDocumentRecordTarget) into
```

```
  memberEndOfPropertyToAssociation
!insert (RecordTargetProperty1,ClinicalDocumentRecordTarget) into
  memberEndOfPropertyToAssociation
!create PatientRole : RoleType
!create PatientRoleProperty1 : Property
!create RecordTargetProperty2 : Property
!create RecordTargetPatientRole : Association
!insert (RecordTarget,RecordTargetProperty2) into TypeToTypedElement
!insert (PatientRole,PatientRoleProperty1) into TypeToTypedElement
!insert (RecordTargetProperty2,RecordTargetPatientRole) into
  memberEndOfPropertyToAssociation
!insert (PatientRoleProperty1,RecordTargetPatientRole) into
  memberEndOfPropertyToAssociation
!create OrganizationProperty2 : Property
!create PatientRoleProperty2 : Property
!create PatientRoleOrganization : Association
!insert (Organization,OrganizationProperty2) into TypeToTypedElement
!insert (PatientRole,PatientRoleProperty2) into TypeToTypedElement
!insert (PatientRoleProperty2,PatientRoleOrganization) into
  memberEndOfPropertyToAssociation
!insert (OrganizationProperty2,PatientRoleOrganization) into
  memberEndOfPropertyToAssociation
!create Patient : EntityType
!create PatientProperty1 : Property
!create PatientRoleProperty3 : Property
!create PatientRolePatient : Association
!insert (PatientRole,PatientRoleProperty3) into TypeToTypedElement
!insert (Patient,PatientProperty1) into TypeToTypedElement
!insert (PatientProperty1,PatientRolePatient) into
  memberEndOfPropertyToAssociation
!insert (PatientRoleProperty3,PatientRolePatient) into
  memberEndOfPropertyToAssociation
!create Birthplace : RoleType
!create BirthplaceProperty1 : Property
!create PatientProperty2 : Property
!insert (Birthplace,BirthplaceProperty1) into TypeToTypedElement
!insert (Patient,PatientProperty2) into TypeToTypedElement
!create BirthplacePatient : Association
!insert (PatientProperty2,BirthplacePatient) into
  memberEndOfPropertyToAssociation
!insert (BirthplaceProperty1,BirthplacePatient) into
  memberEndOfPropertyToAssociation
!create BirthplaceProperty2 : Property
!create Place : EntityType
!create PlaceProperty1 : Property
!create BirthplacePlace : Association
!insert (Birthplace,BirthplaceProperty2) into TypeToTypedElement
!insert (Place,PlaceProperty1) into TypeToTypedElement
!insert (PlaceProperty1,BirthplacePlace) into
  memberEndOfPropertyToAssociation
!insert (BirthplaceProperty2,BirthplacePlace) into
  memberEndOfPropertyToAssociation
!create LanguageCommunication : InfraestructureType
!create LanguageCommunicationProperty1 : Property
!create PatientProperty3 : Property
!create PatientLanguageCommunication : Association
!insert (Patient,PatientProperty3) into TypeToTypedElement
!insert (LanguageCommunication,LanguageCommunicationProperty1) into
  TypeToTypedElement
!insert (PatientProperty3,PatientLanguageCommunication) into
  memberEndOfPropertyToAssociation
```

```
!insert (LanguageCommunicationProperty1,
  PatientLanguageCommunication) into memberEndOfPropertyToAssociation
!create PatientProperty4 : Property
!create Guardian : RoleType
!create GuardianProperty1 : Property
!create GuardianPatient : Association
!insert (Patient,PatientProperty4) into TypeToTypedElement
!insert (Guardian,GuardianProperty1) into TypeToTypedElement
!insert (PatientProperty4,GuardianPatient) into
  memberEndOfPropertyToAssociation
!insert (GuardianProperty1,GuardianPatient) into
  memberEndOfPropertyToAssociation
!create GuardianChoice : Choice
!insert (GuardianChoice,Person) into
  ChoiceToOwnedElementOfChoosableElement
!insert (GuardianChoice,Organization) into
  ChoiceToOwnedElementOfChoosableElement
!create GuardianChoiceProperty1 : Property
!create GuardianProperty2 : Property
!create GuardianGuardianChoice : Association
!insert (GuardianChoice,GuardianChoiceProperty1) into
  TypeToTypedElement
!insert (Guardian,GuardianProperty2) into TypeToTypedElement
!insert (GuardianProperty2,GuardianGuardianChoice) into
  memberEndOfPropertyToAssociation
!insert (GuardianChoiceProperty1,GuardianGuardianChoice) into
  memberEndOfPropertyToAssociation
!create CdaRmim : EntryPoint
!insert (CdaRmim,ClinicalDocument) into
  EntryPointToOwnedElementOfChoosableElement
!create Component : ActRelationshipType
!create ComponentProperty1 : Property
!create ClinicalDocumentProperty4 : Property
!create ClinicalDocumentComponent : Association
!insert (ClinicalDocument,ClinicalDocumentProperty4) into
  TypeToTypedElement
!insert (Component,ComponentProperty1) into TypeToTypedElement
!insert (ClinicalDocumentProperty4,ClinicalDocumentComponent) into
  memberEndOfPropertyToAssociation
!insert (ComponentProperty1,ClinicalDocumentComponent) into
  memberEndOfPropertyToAssociation
!create BodyChoice : Choice
!create BodyChoiceProperty1 : Property
!create ComponentProperty2 : Property
!create ComponentBodyChoice : Association
!insert (Component,ComponentProperty2) into TypeToTypedElement
!insert (BodyChoice,BodyChoiceProperty1) into TypeToTypedElement
!insert (ComponentProperty2,ComponentBodyChoice) into
  memberEndOfPropertyToAssociation
!insert (BodyChoiceProperty1,ComponentBodyChoice) into
  memberEndOfPropertyToAssociation
!create NonXMLBody : ActType
!create StructuredBody : ActType
!insert (BodyChoice,NonXMLBody) into
  ChoiceToOwnedElementOfChoosableElement
!insert (BodyChoice,StructuredBody) into
  ChoiceToOwnedElementOfChoosableElement
!create StructuredBodyProperty1 : Property
!create Component2 : ActRelationshipType
!create Component2Property1 : Property
!create StructuredBodyComponent2 : Association
```

```
!insert (Component2,Component2Property1) into TypeToTypedElement
!insert (StructuredBodyProperty1,StructuredBodyComponent2) into
  memberEndOfPropertyToAssociation
!insert (Component2Property1,StructuredBodyComponent2) into
  memberEndOfPropertyToAssociation
!create Section : ActType
!create SectionProperty1 : Property
!create Component2Property2 : Property
!create SectionComponent2 : Association
!insert (Section,SectionProperty1) into TypeToTypedElement
!insert (Component2,Component2Property2) into TypeToTypedElement
!insert (SectionProperty1,SectionComponent2) into
  memberEndOfPropertyToAssociation
!insert (Component2Property2,SectionComponent2) into
  memberEndOfPropertyToAssociation
!create Component3 : ActRelationshipType
!create Component3Property1 : Property
!create SectionProperty2 : Property
!create SectionComponent3 : Association
!insert (Section,SectionProperty2) into TypeToTypedElement
!insert (Component3,Component3Property1) into TypeToTypedElement
!insert (SectionProperty2,SectionComponent3) into
  memberEndOfPropertyToAssociation
!insert (Component3Property1,SectionComponent3) into
  memberEndOfPropertyToAssociation
!create Entry : ActRelationshipType
!create EntryProperty1 : Property
!create SectionProperty3 : Property
!insert (Section,SectionProperty3) into TypeToTypedElement
!insert (Entry,EntryProperty1) into TypeToTypedElement
!create SectionEntry : Association
!insert (EntryProperty1,SectionEntry) into
  memberEndOfPropertyToAssociation
!insert (SectionProperty3,SectionEntry) into
  memberEndOfPropertyToAssociation
!create ClinicalStatement : Choice
!create ClinicalStatementProperty1 : Property
!create EntryProperty2 : Property
!insert (Entry,EntryProperty2) into TypeToTypedElement
!insert (ClinicalStatement,ClinicalStatementProperty1) into
  TypeToTypedElement
!create ClinicalStatementEntry : Association
!insert (EntryProperty2,ClinicalStatementEntry) into
  memberEndOfPropertyToAssociation
!insert (ClinicalStatementProperty1,ClinicalStatementEntry) into
  memberEndOfPropertyToAssociation
!create Supply : ActType
!create SubstanceAdministration : ActType
!create Observation : ActType
!insert (ClinicalStatement,Supply) into
  ChoiceToOwnedElementOfChoosableElement
!insert (ClinicalStatement,SubstanceAdministration) into
  ChoiceToOwnedElementOfChoosableElement
!insert (ClinicalStatement,Observation) into
  ChoiceToOwnedElementOfChoosableElement
!create SupplyProperty1 : Property
!create Product : ParticipationType
!create ProductProperty1 : Property
!create SupplyProduct : Association
!insert (Supply,SupplyProperty1) into TypeToTypedElement
!insert (Product,ProductProperty1) into TypeToTypedElement
```

```
!insert (SupplyProperty1,SupplyProduct) into
  memberEndOfPropertyToAssociation
!insert (ProductProperty1,SupplyProduct) into
  memberEndOfPropertyToAssociation
!create ManufacturedProduct : RoleType
!create ManufacturedProductProperty1 : Property
!create ProductProperty2 : Property
!create ProductManufacturedProduct : Association
!insert (Product,ProductProperty2) into TypeToTypedElement
!insert (ManufacturedProduct,ManufacturedProductProperty1) into
  TypeToTypedElement
!insert (ProductProperty2,ProductManufacturedProduct) into
  memberEndOfPropertyToAssociation
!insert (ManufacturedProductProperty1,ProductManufacturedProduct)
  into memberEndOfPropertyToAssociation
!create DrugOrOtherMaterial : Choice
!create LabeledDrug : EntityType
!create Material : EntityType
!create DrugOrOtherMaterialProperty1 : Property
!create ManufacturedProductProperty2 : Property
!insert (ManufacturedProduct,ManufacturedProductProperty2) into
  TypeToTypedElement
!insert (DrugOrOtherMaterial,DrugOrOtherMaterialProperty1) into
  TypeToTypedElement
!insert (DrugOrOtherMaterial,LabeledDrug) into
  ChoiceToOwnedElementOfChoosableElement
!insert (DrugOrOtherMaterial,Material) into
  ChoiceToOwnedElementOfChoosableElement
!create ManufacturedProductDrugOrOtherMaterial : Association
!insert (DrugOrOtherMaterialProperty1,
  ManufacturedProductDrugOrOtherMaterial) into
  memberEndOfPropertyToAssociation
!insert (ManufacturedProductProperty2,
  ManufacturedProductDrugOrOtherMaterial) into
  memberEndOfPropertyToAssociation
!create OrganizationProperty3 : Property
!create ManufacturedProductProperty3 : Property
!create ManufacturedProductOrganization : Association
!insert (Organization,OrganizationProperty3) into TypeToTypedElement
!insert (OrganizationProperty3,ManufacturedProductOrganization) into
  memberEndOfPropertyToAssociation
!insert (ManufacturedProductProperty3,
  ManufacturedProductOrganization) into
  memberEndOfPropertyToAssociation
!insert (ManufacturedProduct,ManufacturedProductProperty3) into
  TypeToTypedElement
!create SubstanceAdministrationProperty1 : Property
!create Consumable : ParticipationType
!create ConsumableProperty1 : Property
!create SubstanceAdministrationConsumable : Association
!insert (SubstanceAdministration,SubstanceAdministrationProperty1)
  into TypeToTypedElement
!insert (Consumable,ConsumableProperty1) into TypeToTypedElement
!insert (ConsumableProperty1,SubstanceAdministrationConsumable)
  into memberEndOfPropertyToAssociation
!insert (SubstanceAdministrationProperty1,
  SubstanceAdministrationConsumable) into
  memberEndOfPropertyToAssociation
!create ConsumableProperty2 : Property
!create ManufacturedProductProperty4 : Property
!create ConsumableManufacturedProduct : Association
```

```
!insert (Consumable,ConsumableProperty2) into TypeToTypedElement
!insert (ConsumableProperty2,ConsumableManufacturedProduct) into
  memberEndOfPropertyToAssociation
!insert (ManufacturedProductProperty4,ConsumableManufacturedProduct)
  into memberEndOfPropertyToAssociation
!insert (ManufacturedProduct,ManufacturedProductProperty4) into
  TypeToTypedElement
!create ObservationProperty1 : Property
!create ReferenceRange : ActRelationshipType
!create ReferenceRangeProperty1 : Property
!create ObservationReferenceRange : Association
!insert (Observation,ObservationProperty1) into TypeToTypedElement
!insert (ReferenceRange,ReferenceRangeProperty1) into
  TypeToTypedElement
!insert (ObservationProperty1,ObservationReferenceRange) into
  memberEndOfPropertyToAssociation
!insert (ReferenceRangeProperty1,ObservationReferenceRange) into
  memberEndOfPropertyToAssociation
!create ObservationRange : ActType
!create ObservationRangeProperty1 : Property
!create ReferenceRangeProperty2 : Property
!create ObservationRangeReferenceRange : Association
!insert (ReferenceRange,ReferenceRangeProperty2) into
  TypeToTypedElement
!insert (ObservationRange,ObservationRangeProperty1) into
  TypeToTypedElement
!insert (ReferenceRangeProperty2,ObservationRangeReferenceRange)
  into memberEndOfPropertyToAssociation
!insert (ObservationRangeProperty1,ObservationRangeReferenceRange)
  into memberEndOfPropertyToAssociation
!create ObservationNote : Note
!insert (Observation,ObservationNote) into ElementToNote
!create EntryRelationship : ActRelationshipType
!create ClinicalStatementProperty2 : Property
!create EntryRelationshipProperty1 : Property
!create ClinicalStatementEntryRelationship : Association
!insert (EntryRelationship,EntryRelationshipProperty1) into
  TypeToTypedElement
!insert (ClinicalStatement,ClinicalStatementProperty2) into
  TypeToTypedElement
!insert
(EntryRelationshipProperty1,ClinicalStatementEntryRelationship) into
memberEndOfPropertyToAssociation
!insert (ClinicalStatementProperty2,
  ClinicalStatementEntryRelationship) into
  memberEndOfPropertyToAssociation
!create AuthorProperty3 : Property
!create ClinicalStatementProperty3 : Property
!create ClinicalStatementAuthor : Association
!insert (ClinicalStatementProperty3,ClinicalStatementAuthor) into
  memberEndOfPropertyToAssociation
!insert (AuthorProperty3,ClinicalStatementAuthor) into
  memberEndOfPropertyToAssociation
!insert (Author,AuthorProperty3) into TypeToTypedElement
!insert (ClinicalStatement,ClinicalStatementProperty3) into
  TypeToTypedElement
!insert (StructuredBody,StructuredBodyProperty1) into
  TypeToTypedElement
!create Component3Property2 : Property
!create Section2Component3 : Association
!create Section2Property1 : Property
```

```
!create Section2 : ActType
!insert (Section2,Section2Property1) into TypeToTypedElement
!insert (Section2Property1,Section2Component3) into
  memberEndOfPropertyToAssociation
!insert (Component3Property2,Section2Component3) into
  memberEndOfPropertyToAssociation
!insert (Component3,Component3Property2) into TypeToTypedElement
!create ClinicalStatement2 : Choice
!create ClinicalStatement2Property1 : Property
!create EntryRelationshipProperty2 : Property
!create ClinicalStatement2EntryRelationship : Association
!insert (EntryRelationshipProperty2,
  ClinicalStatement2EntryRelationship) into
  memberEndOfPropertyToAssociation
!insert (ClinicalStatement2Property1,
  ClinicalStatement2EntryRelationship) into
  memberEndOfPropertyToAssociation
!insert (ClinicalStatement2,ClinicalStatement2Property1) into
  TypeToTypedElement
!insert (EntryRelationship,EntryRelationshipProperty2) into
  TypeToTypedElement
!insert (ClinicalStatement2,SubstanceAdministration) into
  ChoiceToOwnedElementOfChoosableElement
!insert (ClinicalStatement2,Observation) into
  ChoiceToOwnedElementOfChoosableElement
!insert (ClinicalStatement2,Supply) into
  ChoiceToOwnedElementOfChoosableElement
!set AssignedAuthorProperty1.upper := - 1
!set AssignedAuthorProperty1.lower := 1
!set AuthorProperty1.upper := - 1
!set AuthorProperty1.lower := 1
!set ClinicalDocumentProperty1.upper := 1
!set ClinicalDocumentProperty1.lower := 1
!set CustodianProperty1.upper := 1
!set CustodianProperty1.lower := 1
!set ClinicalDocumentProperty2.upper := 1
!set ClinicalDocumentProperty2.lower := 1
!set ClinicalDocumentProperty4.upper := 1
!set ClinicalDocumentProperty4.lower := 1
!set ComponentProperty1.upper := 1
!set ComponentProperty1.lower := 1
!set OrganizationProperty1.upper := 1
!set OrganizationProperty1.lower := 0
!set AuthorChoiceProperty1.upper := 1
!set AuthorChoiceProperty1.lower := 0
!set OrganizationProperty2.upper := 1
!set OrganizationProperty2.lower := 0
!set AssignedCustodianProperty1.upper := 1
!set AssignedCustodianProperty1.lower := 1
!set CustodianOrganizationProperty1.upper := 1
!set CustodianOrganizationProperty1.lower := 1
!set RecordTargetProperty1.upper := - 1
!set RecordTargetProperty1.lower := 1
!set ClinicalDocumentProperty3.upper := 1
!set ClinicalDocumentProperty3.lower := 1
!set BodyChoiceProperty1.upper := 1
!set BodyChoiceProperty1.lower := 1
!set MaintainedEntityProperty1.upper := - 1
!set MaintainedEntityProperty1.lower := 0
!set AuthoringDeviceProperty1.upper := 1
!set AuthoringDeviceProperty1.lower := 1
```

```
!set PersonProperty1.upper := 1
!set PersonProperty1.lower := 1
!set PatientRoleProperty1.upper := - 1
!set PatientRoleProperty1.lower := 1
!set PatientProperty1.upper := 1
!set PatientProperty1.lower := 0
!set SectionProperty1.upper := - 1
!set SectionProperty1.lower := 1
!set GuardianProperty1.upper := - 1
!set GuardianProperty1.lower := 0
!set BirthplaceProperty1.upper := 1
!set BirthplaceProperty1.lower := 0
!set SectionProperty2.upper := - 1
!set SectionProperty2.lower := 0
!set GuardianChoiceProperty1.upper := 1
!set GuardianChoiceProperty1.lower := 1
!set PlaceProperty1.upper := 1
!set PlaceProperty1.lower := 1
!set ClinicalStatementProperty1.upper := - 1
!set ClinicalStatementProperty1.lower := 0
!set EntryProperty1.upper := - 1
!set EntryProperty1.lower := 0
!set SectionProperty3.upper := 1
!set SectionProperty3.lower := 1
!set LanguageCommunicationProperty1.upper := - 1
!set LanguageCommunicationProperty1.lower := 0
!set PatientProperty3.upper := 1
!set PatientProperty3.lower := 1
!set OrganizationProperty3.upper := 1
!set OrganizationProperty3.lower := 0
!set ManufacturedProductProperty1.upper := 1
!set ManufacturedProductProperty1.lower := 0
!set ProductProperty2.upper := 1
!set ProductProperty2.lower := 1
!set ProductProperty1.upper := 1
!set ProductProperty1.lower := 0
!set SupplyProperty1.upper := 1
!set SupplyProperty1.lower := 1
!set AuthorProperty3.upper := - 1
!set AuthorProperty3.lower := 0
!set ClinicalStatementProperty3.upper := 1
!set ClinicalStatementProperty3.lower := 1
!set ClinicalStatementProperty2.upper := - 1
!set ClinicalStatementProperty2.lower := 0
!set ManufacturedProductProperty4.upper := 1
!set ManufacturedProductProperty4.lower := 1
!set ObservationRangeProperty1.upper := 1
!set ObservationRangeProperty1.lower := 1
!set DrugOrOtherMaterialProperty1.upper := 1
!set DrugOrOtherMaterialProperty1.lower := 1
!set SubstanceAdministrationProperty1.upper := 1
!set SubstanceAdministrationProperty1.lower := 1
!set ConsumableProperty1.upper := 1
!set ConsumableProperty1.lower := 1
!set ReferenceRangeProperty1.upper := - 1
!set ReferenceRangeProperty1.lower := 0
!set ObservationProperty1.upper := 1
!set ObservationProperty1.lower := 1
!set ObservationRangeProperty1.upper := - 1
!set ObservationRangeProperty1.lower := 0
!set Section2Property1.upper := - 1
```

```
!set Section2Property1.lower := 0
!set Component3Property2.upper := 1
!set Component3Property2.lower := 1
!set AuthorProperty2.upper := - 1
!set AuthorProperty2.lower := 0
!set AssignedAuthorProperty2.upper := - 1
!set AssignedAuthorProperty2.lower := 0
!set CustodianProperty2.upper := - 1
!set CustodianProperty2.lower := 0
!set AssignedCustodianProperty2.upper := - 1
!set AssignedCustodianProperty2.lower := 0
!set AssignedAuthorProperty3.upper := - 1
!set AssignedAuthorProperty3.lower := 0
!set AuthoringDeviceProperty1.upper := 1
!set AuthoringDeviceProperty1.lower := 0
!set MaintainedEntityProperty2.upper := - 1
!set MaintainedEntityProperty2.lower := 0
!set RecordTargetProperty2.upper := - 1
!set RecordTargetProperty2.lower := 0
!set PatientRoleProperty2.upper := - 1
!set PatientRoleProperty2.lower := 0
!set PatientRoleProperty3.lower := 0
!set PatientRoleProperty3.upper := - 1
!set PatientProperty2.upper := 1
!set PatientProperty2.lower := 0
!set BirthplaceProperty2.upper := - 1
!set BirthplaceProperty2.lower := 0
!set PatientProperty4.upper := 0
!set PatientProperty4.lower := 0
!set GuardianProperty2.upper := - 1
!set GuardianProperty2.lower := 0
!set ComponentProperty2.upper := - 1
!set ComponentProperty2.lower := 0
!set StructuredBodyProperty1.upper := 1
!set StructuredBodyProperty1.lower := 1
!set Component2Property1.upper := - 1
!set Component2Property1.lower := 1
!set Component2Property2.upper := - 1
!set Component2Property2.lower := 0
!set Component3Property1.upper := - 1
!set Component3Property1.lower := 0
!set EntryProperty2.upper := - 1
!set EntryProperty2.lower := 0
!set ConsumableProperty2.upper := - 1
!set ConsumableProperty2.lower := 0
!set ManufacturedProductProperty3.upper := - 1
!set ManufacturedProductProperty3.lower := 0
!set ManufacturedProductProperty2.upper := - 1
!set ManufacturedProductProperty2.lower := 0
!set EntryRelationshipProperty1.upper := - 1
!set EntryRelationshipProperty1.lower := 0
!set ReferenceRangeProperty2.upper := - 1
!set ReferenceRangeProperty2.lower := 0
!set ClinicalStatement2Property1.upper := - 1
!set ClinicalStatement2Property1.lower := 0
!set EntryRelationshipProperty2.upper := - 1
!set EntryRelationshipProperty2.lower := 0
!set AuthorChoice.name := 'AuthorChoice'
!set GuardianChoice.name := 'GuardianChoice'
!set DrugOrOtherMaterial.name := 'DrugOrOtherMaterial'
!set BodyChoice.name := 'BodyChoice'
```

```
!set ClinicalStatement.name := 'ClinicalStatement'
!set AssignedAuthor.name := 'AssignedAuthor'
!set Organization.name := 'EntityType'
!set Author.name := 'Author'
!set Custodian.name := 'Custodian'
!set ClinicalDocument.name := 'ClinicalDocument'
!set Component.name := 'Component'
!set AssignedCustodian.name := 'AssignedCustodian'
!set CustodianOrganization.name := 'CustodianOrganization'
!set Person.name := 'Person'
!set AuthoringDevice.name := 'AuthoringDevice'
!set MaintainedEntity.name := 'MaintainedEntity'
!set RecordTarget.name := 'RecordTarget'
!set PatientRole.name := 'PatientRole'
!set Component2.name := 'Component'
!set NonXMLBody.name := 'NonXMLBody'
!set StructuredBody.name := 'StructuredBody'
!set Section.name := 'Section'
!set Patient.name := 'Patient'
!set Guardian.name := 'Guardian'
!set Birthplace.name := 'Birthplace'
!set Place.name := 'Place'
!set Component3.name := 'Component'
!set Entry.name := 'Entry'
!set LanguageCommunication.name := 'LanguageCommunication'
!set Supply.name := 'Supply'
!set Product.name := 'Product'
!set ManufacturedProduct.name := 'ManufacturedProduct'
!set EntryRelationship.name := 'EntryRelationship'
!set LabeledDrug.name := 'LabeledDrug'
!set Material.name := 'Material'
!set ReferenceRange.name := 'ReferenceRange'
!set ObservationRange.name := 'ObservationRange'
!set Consumable.name := 'Consumable'
!set SubstanceAdministration.name := 'Substance'
!set Observation.name := 'Observation'
!set Section2.name := 'Section'
!set CdaRmim.name := 'CDA R-MIM'
!set CdaRmim.identifier := 'POCD_RM000040'
!set CdaRmim.description := 'This RMIM is used to generate the CDA
  specification.'
!set ObservationNote.body := 'Observation.value has cardinality
  [0..*], which does not show up in the Visio representation.'
!set AuthorProperty2.name := 'author'
!set AssignedAuthorProperty1.name := 'assignedAuthor'
!set ClinicalDocumentProperty1.name := 'clinicalDocument'
!set AuthorProperty1.name := 'author'
!set ClinicalDocumentProperty2.name := 'clinicalDocument'
!set CustodianProperty1.name := 'custodian'
!set ClinicalDocumentProperty4.name := 'source-clinicalDocument'
!set ComponentProperty1.name := 'component'
!set AssignedAuthorProperty2.name := 'assignedAuthor'
!set OrganizationProperty1.name := 'scoper-representedOrganization'
!set AssignedAuthorProperty3.name := 'assignedAuthor'
!set AuthorChoiceProperty1.name := 'player-assignedAuthorChoice'
!set OrganizationProperty2.name := 'scoper-providerOrganization'
!set PatientRoleProperty2.name := 'patientRole'
!set CustodianProperty2.name := 'custodian'
!set AssignedCustodianProperty1.name := 'assignedCustodian'
!set AssignedCustodianProperty2.name := 'assignedCustodian'
!set CustodianOrganizationProperty1.name := 'scoper-
```

```
  representedCustodianOrganization'
!set ClinicalDocumentProperty3.name := 'clinicalDocument'
!set RecordTargetProperty1.name := 'recordTarget'
!set ComponentProperty2.name := 'component'
!set BodyChoiceProperty1.name := 'target-bodyChoice'
!set MaintainedEntityProperty1.name := 'asMaintainedEntity'
!set AuthoringDeviceProperty1.name := 'player-authoringDevice'
!set MaintainedEntityProperty2.name := 'maintainedEntity'
!set PersonProperty1.name := 'scoper-maintainingPerson'
!set RecordTargetProperty2.name := 'recordTarget'
!set PatientRoleProperty1.name := 'patientRole'
!set PatientRoleProperty3.name := 'patientRole'
!set PatientProperty1.name := 'player-patient'
!set StructuredBodyProperty1.name := 'source-structuredBody'
!set Component2Property1.name := 'component'
!set Component2Property2.name := 'component'
!set SectionProperty1.name := 'target-section'
!set Component3Property1.name := 'component'
!set SectionProperty2.name := 'source-section'
!set PatientProperty4.name := 'scoper-patient'
!set GuardianProperty1.name := 'guardian'
!set GuardianChoiceProperty1.name := 'player-guardianChoice'
!set GuardianProperty2.name := 'guardian'
!set PatientProperty2.name := 'scoper-patient'
!set BirthplaceProperty1.name := 'birthplace'
!set BirthplaceProperty2.name := 'birthplace'
!set PlaceProperty1.name := 'player-place'
!set LanguageCommunicationProperty1.name := 'languageCommunication'
!set PatientProperty3.name := 'patient'
!set EntryProperty2.name := 'entry'
!set ClinicalStatementProperty1.name := 'target-clinicalStatement'
!set SectionProperty3.name := 'source-section'
!set EntryProperty1.name := 'entry'
!set ManufacturedProductProperty3.name := 'manufacturedProduct'
!set OrganizationProperty3.name := 'scoper-manufacturerOrganization'
!set ProductProperty2.name := 'product'
!set ManufacturedProductProperty1.name := 'manufacturedProduct'
!set ProductProperty1.name := 'product'
!set SupplyProperty1.name := 'supply'
!set AuthorProperty3.name := 'author'
!set ClinicalStatementProperty3.name := 'clinicalStatement'
!set ManufacturedProductProperty4.name := 'manufacturedProduct'
!set ConsumableProperty2.name := 'consumable'
!set EntryRelationshipProperty1.name := 'entryRelationship'
!set ClinicalStatementProperty2.name := 'source-clinicalStatement'
!set ReferenceRangeProperty2.name := 'referenceRange'
!set ObservationRangeProperty1.name := 'target-observationRange'
!set DrugOrOtherMaterialProperty1.name := 'player-drugOrOtherMaterial'
!set ManufacturedProductProperty2.name :=
  'manufacturedDrugOrOtherMaterial'
!set SubstanceAdministrationProperty1.name :=
  'substanceAdministration'
!set ConsumableProperty1.name := 'consumable'
!set ReferenceRangeProperty1.name := 'referenceRange'
!set ObservationProperty1.name := 'source-observation'
!set Component3Property2.name := 'component'
!set Section2Property1.name := 'target-section'
!set ClinicalStatement2Property1.name := 'target-clinicalStatement'
!set EntryRelationshipProperty2.name := 'entryRelationship'
```

ANEXO C. Instaciación USE de un subconjunto del esquema Scheduling

```
!create classCodeActApp : Property
!create moodCodeActApp : Property
!create idActApp : Property
!create codeActApp : Property
!create textActApp : Property
!create statusCodeActApp : Property
!create effectiveTimeActApp : Property
!create priorityCodeActApp : Property
!create confidentialityCodeActApp : Property
!create SchedulingDMIM : EntryPoint
!create ActAppointment : ActType
!create ActAppointmentProperty1 : Property
!create ActAppointmentSubject : Association
!create SubjectProperty1 : Property
!create Subject : ParticipationType
!create SubjectProperty2 : Property
!create R_Patient : CMET
!create R_PatientProperty1 : Property
!create SubjectR_Patient : Association
!insert (SchedulingDMIM,ActAppointment) into
  EntryPointToOwnedElementOfChoosableElement
!insert (ActAppointment,ActAppointmentProperty1) into
  TypeToTypedElement
!insert (Subject,SubjectProperty1) into TypeToTypedElement
!insert (ActAppointmentProperty1,ActAppointmentSubject) into
  memberEndOfPropertyToAssociation
!insert (SubjectProperty1,ActAppointmentSubject) into
  memberEndOfPropertyToAssociation
!insert (SubjectProperty2,SubjectR_Patient) into
  memberEndOfPropertyToAssociation
!insert (R_PatientProperty1,SubjectR_Patient) into
  memberEndOfPropertyToAssociation
!insert (R_Patient,R_PatientProperty1) into TypeToTypedElement
!insert (Subject,SubjectProperty2) into TypeToTypedElement
!create ActAppointmentProperty2 : Property
!create ActAppointmentPerformer : Association
!create PerformerProperty1 : Property
!create Performer : ParticipationType
!create PerformerR_AssignedPerson : Association
!create R_AssignedPersonProperty1 : Property
!create R_AssignedPerson : CMET
!create PerformerProperty2 : Property
!insert (ActAppointmentProperty2,ActAppointmentPerformer) into
  memberEndOfPropertyToAssociation
!insert (PerformerProperty1,ActAppointmentPerformer) into
  memberEndOfPropertyToAssociation
!insert (Performer,PerformerProperty1) into TypeToTypedElement
!insert (Performer,PerformerProperty2) into TypeToTypedElement
!insert (R_AssignedPerson,R_AssignedPersonProperty1) into
  TypeToTypedElement
!insert (R_AssignedPersonProperty1,PerformerR_AssignedPerson) into
  memberEndOfPropertyToAssociation
!insert (PerformerProperty2,PerformerR_AssignedPerson) into
  memberEndOfPropertyToAssociation
!insert (ActAppointment,ActAppointmentProperty2) into
  TypeToTypedElement
```

```
!insert (ActAppointment,classCodeActApp) into
  ClassToOwnedAttributeofProperty
!insert (ActAppointment,moodCodeActApp) into
  ClassToOwnedAttributeofProperty
!insert (ActAppointment,idActApp) into ClassToOwnedAttributeofProperty
!insert (ActAppointment,codeActApp) into
  ClassToOwnedAttributeofProperty
!insert (ActAppointment,textActApp) into
  ClassToOwnedAttributeofProperty
!insert (ActAppointment,statusCodeActApp) into
  ClassToOwnedAttributeofProperty
!insert (ActAppointment,effectiveTimeActApp) into
  ClassToOwnedAttributeofProperty
!insert (ActAppointment,priorityCodeActApp) into
  ClassToOwnedAttributeofProperty
!insert (ActAppointment,confidentialityCodeActApp) into
  ClassToOwnedAttributeofProperty
!create typeCodeSubject : Property
!create timeSubject : Property
!create modeCodeSubject : Property
!insert (Subject,typeCodeSubject) into ClassToOwnedAttributeofProperty
!insert (Subject,timeSubject) into ClassToOwnedAttributeofProperty
!insert (Subject,modeCodeSubject) into ClassToOwnedAttributeofProperty
!create typeCodePerformer : Property
!create functionCodePerformer : Property
!create timePerformer : Property
!create modeCodePerformer : Property
!create performIndPerformer : Property
!create substitutionConditionCodePerformer : Property
!insert (Performer,typeCodePerformer) into
  ClassToOwnedAttributeofProperty
!insert (Performer,functionCodePerformer) into
  ClassToOwnedAttributeofProperty
!insert (Performer,timePerformer) into ClassToOwnedAttributeofProperty
!insert (Performer,modeCodePerformer) into
  ClassToOwnedAttributeofProperty
!insert (Performer,performIndPerformer) into
  ClassToOwnedAttributeofProperty
!insert (Performer,substitutionConditionCodePerformer) into
  ClassToOwnedAttributeofProperty
!set SchedulingDMIM.name := 'Scheduling DMIM'
!set SchedulingDMIM.identifier := 'PRSC_DM000000UV'
!set SchedulingDMIM.description := 'Domain model for Scheduling
  Appointments'
!set ActAppointment.name := 'ActAppointment'
!set Subject.name := 'Subject'
!set Performer.name := 'Performer'
!set R_Patient.rootClassCode := 'PAT'
!set R_Patient.attributionLevel := 'identified/confirmable'
!set R_Patient.identifier := 'COCT_MT050002UV'
!set R_Patient.name := 'R_Patient'
!set R_AssignedPerson.rootClassCode := 'ASSIGNED'
!set R_AssignedPerson.attributionLevel := 'universal'
!set R_AssignedPerson.identifier := 'COCT_MT090100UV'
!set R_AssignedPerson.name := 'R_AssignedPerson'
!create Patient : RoleType
!insert (R_Patient,Patient) into CMETToMainElementOfChoosableElement
!insert (R_Patient,Patient) into
  ownerOfCMETToOwnedElementOfChoosableElement
!create AssignedPerson : RoleType
!insert (R_AssignedPerson,AssignedPerson) into
```

```
  ownerOfCMETToOwnedElementOfChoosableElement
!insert (R_AssignedPerson,AssignedPerson) into
  CMETToMainElementOfChoosableElement
!set AssignedPerson.name := 'AssignedPerson'
!set Patient.name := 'Patient'
!set classCodeActApp.upper := 1
!set classCodeActApp.lower := 1
!set classCodeActApp.name := 'classCode'
!set moodCodeActApp.upper := 1
!set moodCodeActApp.lower := 1
!set moodCodeActApp.name := 'moodCode'
!set idActApp.upper := 1
!set idActApp.lower := 1
!set idActApp.name := 'id'
!set codeActApp.upper := 1
!set codeActApp.lower := 0
!set codeActApp.name := 'code'
!set textActApp.upper := 1
!set textActApp.lower := 0
!set textActApp.name := 'text'
!set statusCodeActApp.upper := 1
!set statusCodeActApp.lower := 0
!set statusCodeActApp.name := 'statusCode'
!set effectiveTimeActApp.upper := 1
!set effectiveTimeActApp.lower := 0
!set effectiveTimeActApp.name := 'effectiveTime'
!set priorityCodeActApp.upper := - 1
!set priorityCodeActApp.lower := 0
!set priorityCodeActApp.name := 'priorityCode'
!set confidentialityCodeActApp.upper := - 1
!set confidentialityCodeActApp.lower := 0
!set confidentialityCodeActApp.name := 'confidentialityCode'
!set typeCodeSubject.upper := 1
!set typeCodeSubject.lower := 1
!set typeCodeSubject.name := 'typeCode'
!set timeSubject.upper := 1
!set timeSubject.lower := 0
!set timeSubject.name := 'time'
!set modeCodeSubject.upper := 1
!set modeCodeSubject.lower := 0
!set modeCodeSubject.name := 'modeCode'
!set typeCodePerformer.upper := 1
!set typeCodePerformer.lower := 1
!set typeCodePerformer.name := 'typeCode'
!set functionCodePerformer.upper := 1
!set functionCodePerformer.lower := 0
!set functionCodePerformer.name := 'functionCode'
!set timePerformer.upper := 1
!set timePerformer.lower := 0
!set timePerformer.name := 'time'
!set modeCodePerformer.upper := 1
!set modeCodePerformer.lower := 0
!set modeCodePerformer.name := 'modeCode'
!set performIndPerformer.upper := 1
!set performIndPerformer.lower := 0
!set performIndPerformer.name := 'performInd'
!set substitutionConditionCodePerformer.upper := 1
!set substitutionConditionCodePerformer.lower := 0
!set substitutionConditionCodePerformer.name :=
  'substitutionConditionCode'
!set SubjectProperty1.upper := - 1
```

```
!set SubjectProperty1.lower := 0
!set ActAppointmentProperty1.upper := 1
!set ActAppointmentProperty1.lower := 1
!set SubjectProperty2.upper := - 1
!set SubjectProperty2.lower := 0
!set R_PatientProperty1.upper := - 1
!set R_PatientProperty1.lower := 1
!set ActAppointmentProperty2.upper := 1
!set ActAppointmentProperty2.lower := 1
!set PerformerProperty1.upper := - 1
!set PerformerProperty1.lower := 0
!set R_AssignedPersonProperty1.upper := - 1
!set R_AssignedPersonProperty1.lower := 0
!set PerformerProperty2.upper := - 1
!set PerformerProperty2.lower := 0
!set ActAppointmentProperty1.name := 'actAppointment'
!set SubjectProperty1.name := 'subject'
!set R_PatientProperty1.name := 'patient'
!set SubjectProperty2.name := 'subject'
!set ActAppointmentProperty2.name := 'actAppointment'
!set PerformerProperty1.name := 'performer'
!set PerformerProperty2.name := 'performer'
!set R_AssignedPersonProperty1.name := 'assignedPerson'
```

ANEXO D. Módulo ATL con las reglas de transformación de HL7 a UML

```
-- @nsURI UML2=http://www.eclipse.org/uml2/3.0.0/UML
-- @path HL7=/HL7ToXMI/HL7_Metamodel.ecore

module HL7ToXMI;
create OUT : UML2 from IN : HL7, DT : UML2, PRO : UML2;

--Primera regla que se ejecuta
rule EntryPoint {
  from
    ep: HL7!EntryPoint
  to
    --se crea el modelo
    m: UML2!Model (
      name <- ep.identifier,
      packageImport <- pi
    ),
    pi: UML2!PackageImport (
      importedPackage <- UML2!Package.allInstancesFrom('DT')
       ->select(p | p.name = 'DataTypes').first()
    ),
    --se crea la clase EntryPoint con un atributo description
    c: UML2!Class (
      ownedAttribute <- pr1,
      isAbstract <- true
    ),
    pr1: UML2!Property (
      name <- 'description',
      upper <- 1,
      lower <- 0,
      isReadOnly <- true,
      default <- ep.description
    ),
    --se crea la asociación entre el EntryPoint y la clase foco
    pr2: UML2!Property(
      upper <- 1,
      lower <- 0,
      type <- c,
      name <- c.name
    ),
    pr3: UML2!Property(
      upper <- -1,
      lower <- 0
    ),
    a: UML2!Association (
      ownedEnd <- Set{pr2, pr3}
    )
  do {
    --En los casos donde no se disponga del nombre del EntryPoint se
    usa el id como nombre de la clase EntryPoint
    if (not ep.name.oclIsUndefined()) c.name <- ep.name;
    else c.name <- ep.identifier;

    --Se aplica el profile al modelo
    m.applyProfile(UML2!Profile.allInstancesFrom('PRO')
     ->select(p | p.name = 'Profile').first());
```

```
    m.packagedElement.add(c);
    m.packagedElement.add(a);

    --Se aplica el estereotipo 'EntryPoint'
    c.applyStereotype(c.getApplicableStereotype(
    'Profile::EntryPoint'));

    for (c in HL7!Class.allInstancesFrom('IN')) {
      thisModule.Class(m, c);
    }
    for (cm in HL7!CMET.allInstancesFrom('IN')) {
      thisModule.CMET(m, cm);
    }
    for (ch in HL7!Choice.allInstancesFrom('IN')) {
      thisModule.Choice(m, ch);
    }
    for (a in HL7!Association.allInstancesFrom('IN')) {
      thisModule.Association(m, a);
    }
    pr1.type <- UML2!PrimitiveType.allInstancesFrom('DT')
      ->select(pt | pt.name = 'String').first();

    --Se asigna el tipo a la clase foco del modelo, esto se tiene
     que hacer después de haber creado las clases
    pr3.type <- UML2!Class.allInstancesFrom('OUT')->
     select(c | c.name = ep.choosableElement.name).first();
  }
}

--Se crea una clase, se estereotipa y se le asignan sus atributos y
comentarios
rule Class(m: UML2!Model, c1: HL7!Class) {
  to
    c2: UML2!Class (
      name <- c1.name
    )
  do {
    m.packagedElement.add(c2);
    --Se estereotipa la clase según su tipo
    if (c1.oclIsTypeOf(HL7!ActType)) {
      c2.applyStereotype(c2.getApplicableStereotype(
       'Profile::Act'));
    }
    else if (c1.oclIsTypeOf(HL7!ActRelationshipType)) {
      c2.applyStereotype(c2.getApplicableStereotype(
       'Profile::ActRelationship'));
    }
    else if (c1.oclIsTypeOf(HL7!ParticipationType)) {
      c2.applyStereotype(c2.getApplicableStereotype(
       'Profile::Participation'));
    }
    else if (c1.oclIsTypeOf(HL7!RoleType)) {
      c2.applyStereotype(c2.getApplicableStereotype(
       'Profile::Role'));
    }
    else if (c1.oclIsTypeOf(HL7!RoleLinkType)) {
      c2.applyStereotype(c2.getApplicableStereotype(
       'Profile::RoleLink'));
    }
    else if (c1.oclIsTypeOf(HL7!EntityType)) {
      c2.applyStereotype(c2.getApplicableStereotype(
```

```
        'Profile::Entity'));
    }
    else if (c1.oclIsTypeOf(HL7!InfraestructureType)){
      c2.applyStereotype(c2.getApplicableStereotype('
       Profile::Infraestructure'));
    }
    for (a in c1.ownedAttribute) {
      thisModule.Attribute(c2, a);
    }
    for (n in c1.note) {
      thisModule.ClassNote(c2, n);
    }
  }
}

--Se crea un property y se le asignan sus atributos y comentarios
rule Attribute (c: UML2!Class, p: HL7!Property) {
  to
    pr: UML2!Property (
      name <- p.name,
      upper <- p.upper,
      lower <- p.lower,
      type <- UML2!DataType.allInstancesFrom('DT')->
       select(dt | dt.name = p.type.name).first()
    )
  do {
    --Si el atributo es classCode o typeCode se pone como readOnly
    if (p.name = 'classCode' or p.name = 'typeCode') {
      pr.isReadOnly <- true;
    }
    c.ownedAttribute.add(pr);
    for (n in p.note) {
      thisModule.AttributeNote(pr, n);
    }
    --Se asignan en el valor default: codingStrength, mnemonic,
     domainName y valor por defecto
    if (not p.codingStrength.oclIsUndefined()) {
      pr.default <- p.codingStrength;
    }
    if (not p.domainName.oclIsUndefined()) {
      if (not p.codingStrength.oclIsUndefined()) {
        pr.default <- pr.default.concat(', ').concat(p.domainName);
      }
      else pr.default <- p.domainName;
    }
    if (not p.codeSystemName.oclIsUndefined()) {
      if (not p.codingStrength.oclIsUndefined()) {
        pr.default <- pr.default.concat(',
         ').concat(p.codeSystemName);
      }
      else pr.default <- p.codeSystemName;
    }
    if (not p.mnemonic.oclIsUndefined()) {
      if (not p.codingStrength.oclIsUndefined() or not
       p.domainName.oclIsUndefined()) {
        pr.default <- pr.default.concat(':').concat(p.mnemonic);
      }
      else pr.default <- p.mnemonic;
    }
    if (not p.default.oclIsUndefined()) {
      if (not p.codingStrength.oclIsUndefined() or not
```

```
        p.domainName.oclIsUndefined()
        or not p.mnemonic.oclIsUndefined()) {
         pr.default <- pr.default.concat(', default =
          "').concat(p.default).concat('"');
       }
       else pr.default <- p.default;
    }
    if (not pr.default.oclIsUndefined()) {
      pr.default <- '{' + pr.default + '}';
    }
  }
}

--Se crea una nota y se asigna a la clase afectada
rule ClassNote(cl: UML2!Class, n: HL7!Note) {
  to
    co: UML2!Comment
  do {
    if (n.type = 'Usage') {
      co.body <- '<Usage> ' + n.body;
    }
    else if (n.type = 'Definition') {
      co.body <- '<Definition> ' + n.body;
    }
    cl.ownedComment.add(co);
  }
}

--Se crea una nota y se asigna al atributo afectado
rule AttributeNote(pr: UML2!Property, n: HL7!Note) {
  to
    co: UML2!Comment
  do {
    if (n.type = 'Usage') {
      co.body <- '<Usage> ' + n.body;
    }
    else if (n.type = 'Definition') {
      co.body <- '<Definition> ' + n.body;
    }
    pr.ownedComment.add(co);
  }
}

--Se crea una asociación
rule Association(m: UML2!Model, a1: HL7!Association) {
  using {
    memberNames : Set(String) = a1.memberEnd->collect(me |
     me.type.name);
    firstMemberName : String = memberNames.first();
    secondMemberName : String = memberNames.last();
    firstMemberType : UML2!Class = UML2!Class.allInstancesFrom('OUT')
     ->select(c | c.name = firstMemberName).first();
    secondMemberType : UML2!Class =
     UML2!Class.allInstancesFrom('OUT')->
     select(c | c.name = secondMemberName).first();
  }
  to
    p1: UML2!Property(
      upper <- a1.memberEnd.first().upper,
      lower <- a1.memberEnd.first().lower,
      name <- a1.memberEnd.first().name,
```

```
      type <- firstMemberType
    ),
    p2: UML2!Property(
      upper <- a1.memberEnd.last().upper,
      lower <- a1.memberEnd.last().lower,
      name <- a1.memberEnd.last().name,
      type <- secondMemberType
    ),
    a2: UML2!Association (
      ownedEnd <- Set{p1, p2}
    )
  do {
    m.packagedElement.add(a2);
  }
}

--Se crea un Choice
rule Choice(m: UML2!Model, ch: HL7!Choice) {
  to
    c1: UML2!Class (
      name <- ch.name,
      isAbstract <- true
    )
  do {
    m.packagedElement.add(c1);
    c1.applyStereotype(c1.getApplicableStereotype(
     'Profile::Choice'));
    for (n in ch.note) {
      thisModule.ClassNote(c1, n);
    }
    for (oe in ch.ownedElement) {
      thisModule.ChoiceHierarchy(c1,
       UML2!Class.allInstancesFrom('OUT')->
       select(c | c.name = oe.name).first());
    }
  }
}

--Se crea una generalización que se da dentro de un choice
rule ChoiceHierarchy(parentClass: UML2!Class, childClass: UML2!Class)
{
  to
    g: UML2!Generalization (
      general <- parentClass,
      specific <- childClass
    )
  do {
    childClass.generalization.add(g);
  }
}

--Se crea un CMET
rule CMET(m: UML2!Model, cm: HL7!CMET) {
  to
    c1: UML2!Class (
      name <- cm.name,
      ownedAttribute <- pr
    ),
    pr: UML2!Property (
      name <- 'identifier',
      upper <- 1,
```

```
      lower <- 1,
      type <- UML2!PrimitiveType.allInstancesFrom('DT')
       ->select(pt | pt.name = 'String').first(),
      isReadOnly <- true,
      default <- cm.identifier
    )
  do {
    m.packagedElement.add(c1);
    c1.applyStereotype(c1.getApplicableStereotype('Profile::cmet'));
    if (cm.mainClassType = 'Act') {
      c1.applyStereotype(c1.getApplicableStereotype('Profile::Act'));
    }
    else if (cm.mainClassType = 'ActRelationship') {
      c1.applyStereotype(c1.getApplicableStereotype('
       Profile::ActRelationship'));
    }
    else if (cm.mainClassType = 'Participation') {
      c1.applyStereotype(c1.getApplicableStereotype(
       'Profile::Participation'));
    }
    else if (cm.mainClassType = 'Role') {
      c1.applyStereotype(c1.getApplicableStereotype('Profile::Role'));
    }
    else if (cm.mainClassType = 'RoleLink') {
      c1.applyStereotype(c1.getApplicableStereotype('
       Profile::RoleLink'));
    }
    else if (cm.mainClassType = 'Entity') {
      c1.applyStereotype(c1.getApplicableStereotype(
       'Profile::Entity'));
    }
    else if (cm.mainClassType = 'Infraestructure'){
      c1.applyStereotype(c1.getApplicableStereotype(
       'Profile::Infraestructure'));
    }
    for (n in cm.note) {
      thisModule.ClassNote(c1, n);
    }
  }
}
```

BIBLIOGRAFÍA

Fuentes citadas en el documento

[AccBas] Esquema HL7 A_AccountPayee basic del ballot publicado en septiembre de 2010 http://www.hl7.org/v3ballot/html/domains/uvct/uvct_do_cmets.html#COCT_RM110202UV [fecha de consulta: 2 de diciembre de 2010]

[AccUn] Esquema HL7 A_AccountPayee universal del ballot publicado en septiembre de 2010 http://www.hl7.org/v3ballot/html/domains/uvct/uvct_do_cmets.html#COCT_RM110200UV [fecha de consulta: 2 de diciembre de 2010]

[ATLw] Página web del proyecto ATL http://www.eclipse.org/atl/ [fecha de consulta: 13 de abril de 2010]

[Ball10] Ballot publicado en septiembre de 2010 http://www.hl7.org/v3ballot2010sep/html/ [fecha de consulta: 26 de abril de 2011]

[Biehl] Matthias Biehl. *Literature Study on Model Transformations*, 2010.

[CDA] Esquema Clinical Document Architecture del ballot de septiembre de 2009 http://www.hl7.org/v3ballot2009sep/html/infrastructure/cda/graphics/L-POCD_RM000040.gif [fecha de consulta: 3 de diciembre de 2010]

[CharUn] Esquema HL7 A_Charge universal del ballot publicado en septiembre de 2010 http://www.hl7.org/v3ballot/html/domains/uvct/uvct_do_cmets.html#COCT_RM400000UV [fecha de consulta: 25 de noviembre de 2010]

[CoveUn] Esquema HL7 A_Coverage basic del ballot publicado en septiembre de 2010 http://www.hl7.org/v3ballot/html/domains/uvct/uvct_do_cmets.html#COCT_RM510005UV [fecha de consulta: 25 de noviembre de 2010]

[EclFor] Mensaje del foro de Eclipse donde se puede encontrar la clase que trabaja con la API de Eclipse para leer y escribir modelos UML http://dev.eclipse.org/newslists/news.eclipse.tools.uml2/msg00833.html [fecha de consulta: 18 de mayo de 2011]

[EclM2M] Página web del proyecto Eclipse M2M http://www.eclipse.org/m2m [fecha de consulta: 28 de febrero de 2011]

[EclMod] Paquete Modeling de la plataforma Eclipse http://www.eclipse.org/modeling/ [fecha de consulta: 28 de febrero de 2011]

[Hl7Ball] Ballots de HL7 publicados hasta la fecha http://www.hl7.org/v3ballot/html/ [fecha de consulta: 15 de junio de 2011]

[hl7Wiki] Página de la wiki de HL7 donde se discute la transformación de los esquemas HL7 a UML http://wiki.hl7.org/index.php?title=RMIM_Diagram_Representation [fecha de consulta: 4 de marzo de 2011]

[HyperMw] Página web del proyecto HyperModel http://www.xmlmodeling.com/hypermodel [fecha de consulta: 4 de marzo de 2011]

[IJsal] Comparativa de salarios http://salarios.infojobs.net [fecha de consulta: 3 de junio de 2011]

[MDHTw] Página web del proyecto MDHT https://www.projects.openhealthtools.org/sf/projects/mdht/ [fecha de consulta: 4 de marzo de 2011]

[MDraw] Página web de la herramienta Magic Draw http://www.magicdraw.com/ [fecha de consulta: 7 de junio de 2011]

[OclEsp] Especificación formal de OCL http://www.omg.org/spec/OCL/ [fecha de consulta: 28 de abril de 2011]

[RIM10] RIM del ballot publicado en septiembre de 2010 http://www.hl7.org/v3ballot2010sep/html/welcome/environment/index.htm

[fecha de consulta: 26 de agosto de 2010]
[Sched] D-MIM del dominio de Scheduling del ballot de septiembre de 2009 http://www.hl7.org/v3ballot2009sep/html/domains/uvsc/editable/PRSC_DM000000UV.htm [fecha de consulta: 3 de diciembre de 2010]
[TDBall] Definición de los tipos de datos en el ballot de septiembre de 2010 http://www.hl7.org/v3ballot2010sep/html/infrastructure/itsuml/datatypes-its-uml.htm#Intro [fecha de consulta: 13 de diciembre de 2010]
[UmlEsp] Especificación formal de UML http://www.omg.org/spec/UML/ [fecha de consulta: 21 de octubre de 2010]
[Usew] Página web de la herramienta USE http://www.db.informatik.uni-bremen.de/projects/USE/ [fecha de consulta: 5 de octubre de 2010]
[Vogel] Tutorial creado por Lars Vogel acerca de tratar ficheros XML usando la API de Eclipse http://www.vogella.de/articles/EclipseEMFPersistence/article.html [fecha de consulta: 10 de mayo de 2011]
[XmiEsp] Espeficiación formal de XMI http://www.omg.org/spec/XMI/ [fecha de consulta: 3 de mayo de 2011]

Otras referencias consultadas

[Benson] Tim Benson. *Principles of Health Interoperability HL7 and SNOMED*, 2010. Springer.
[CFST] Dolors Costal, Xavier Franch, M.Ribera Sancho y Ernest Teniente. *Enginyeria del software – Especificació*, 2005. Edicions UPC.
[gforgew] Página web de GForge, repositorio de proyectos relacionados con HL7 http://gforge.hl7.org/gf/ [fecha de consulta: 22 de febrero de 2011]
[Hl7Esw] Página web de HL7 España http://www.hl7spain.org [fecha de consulta: 5 de junio de 2011]
[Hl7w] Página web del estándar HL7 http://www.hl7.org [fecha de consulta: 5 de junio de 2011]
[OHTw] Página web de OpenHealthTools. Contiene varios proyectos relacionados con HL7 http://www.openhealthtools.org/ [fecha de consulta: 22 de febrero de 2011]
[Olive] Antoni Olivé. *Conceptual Modeling Of Information Systems*, 2007. Springer.
[RJB] James Rumbaugh, Ivar Jacobson y Grady Booch. *The Unified Modeling Language Reference Manual*, 1999. Addison-Wesley.